Chapter 4

Spring 2010 Edition

Logic Minimization Using Karnaugh Maps

A *SUMMARY* of what you learned in Chapter 3:

1- Switching algebra
   - Some frequently used switching theorems.
   - Principle of duality.
   - Bubble-to-bubble notation.
   - Logic simplification.

2- Concepts of minterms and maxterms.

3- Two more representations for logic functions: canonical SOP and canonical POS.

4- NOT, AND and OR gates are sufficient to realize any logic function.

5- Two-input NAND gates (and also two-input NOR gates) are sufficient to realize any logic function.

6- How to reach exact logic circuits from logic expressions and vice versa.

7- How to obtain truth tables from logic circuits or logic expressions.

Introduction

Remember that a logic function, in other words a truth table, may be represented by different logic expressions. But different logic expressions have different realizations with different numbers of gates and/or different-size gates[1] and therefore different costs. We utilized switching algebra to simplify logic expressions in Chapter 3. For example, in Example 9 of Chapter 3, expression (a . b) + (a . b . c'. d) + (a . b . d . e') was manipulated using switching algebra to eventually reach a . b. The original expression needs 1 two-input AND gate, 2 four-input AND gates and 1 three-input OR gate, while the simplified expression needs only 1 two-input AND gate: a significant difference between two realizations of the same function. We, of course, will not hesitate to choose the second one!

As another example let's use switching algebra to simplify the following expression, hence reduce the realization cost. The theorems used in this chapter were introduced in Chapter 3.

$Y = \sum_{A, B, C} (2, 3, 6, 7) = (A' . B . C') + (A . B . C') + (A'. B . C) + (A . B . C)$

Apply T10-L to the first two p-terms and also to the last two p-terms: $Y = B . C' + B . C$

Apply T10-L again to the two resulting p-terms: $Y = B$

Interestingly, the realization cost of Y has been reduced from 4 three-input AND gates and 1 four-input OR gate to NOTHING! So, logic simplification (or ideally, logic minimization) is not an issue that we may afford to overlook in logic design. Switching algebra is the first tool targeted at this goal. Although switching algebra is very flexible, it is not always that simple to manually identify the right switching theorems and apply them properly to logic expressions in hand. In this chapter a graphical representation for logic functions called a *Karnaugh map* is introduced to facilitate *straightforward* and *manual*

[1] By "size" we mean the number of inputs. Keep in mind that "size" may be used in a different context as well: two different-size two-input NAND gates, for example, have different-size *transistors* to tailor their electrical characteristics to our specific needs. This notion of "size" is beyond the scope of this book.

minimization of functions with *a few* variables[2]; hence manual design of medium-size digital circuits. The design procedure usually begins with translating the problem description in natural language into a truth table as explained in Chapter 1. This minimization procedure is able to obtain a *minimal* (2-level) SOP for any switching function. By a minimal SOP we mean a SOP expression (AND-OR logic) with as few product terms as possible. If there are two or more SOP expressions meeting this criterion, the minimal SOP is the one with the lowest number of literals. The first constraint guarantees that the number of product terms or the number of inputs of the second-level OR gate is minimal. And the second constraint guarantees that the number of first-level AND-gate inputs is minimal. Two or more SOP expressions, if any, with the same number of p-terms may also happen to have the same number of literals; hence there might be more than one minimal SOP expression for the same function, as we will see in this chapter.

Similar concepts are valid for POS expressions (OR-AND logic) as well; however, in order to avoid possible confusion, we first consider minimal SOP expressions and then generalize the concepts already developed to also reach minimal POS expressions.

Karnaugh Maps and Combining Single Cells

A 2-, 3-, or 4-variable[3] Karnaugh map (or *K-map* for short) is in fact a two-dimensional version of the corresponding truth table; so that input variables and all of their possible combinations are now split and seated on both X and Y directions as shown in Figure 1*a* for a three-variable K-map. Compare this K-map with the three-variable truth-table shown in Figure 1*b*, where all the input bit patterns are listed in one column only. This two-dimensional coding partitions a K-map into as many cells as the rows that exist in a same-size truth table; so that each cell in a K-map corresponds to one row in a same-size truth table and vice versa. In other words, each cell in a K-map is still assigned a unique bit pattern, which is the vertical coordinates followed by the horizontal coordinates (or simply coordinates) of that cell. For example, the coordinates of the upper-right corner cell in the map of Figure 1*a* are A B C = 100, making this cell equivalent to row 4 of the truth table in Figure 1*b*. (Row 4 has the same input combination, A B C = 100.)

C \ AB	00	01	11	**10**
0	0	2	6	**4**
1	1	3	7	5

(a)

Row	A B C	y
0	0 0 0	
1	0 0 1	
2	0 1 0	
3	0 1 1	
4	**1 0 0**	
5	1 0 1	
6	1 1 0	
7	1 1 1	

(b)

Figure 1. (a) 3-variable K-map, (b) 3-variable truth table

Cell numbers in K-maps are the counterparts of row numbers in truth tables: similar to row numbers in truth tables, a cell number in a K-map is the decimal equivalent for the binary coordinates of that cell. Cell numbers are shown inside the cells in Figure 1*a*. For example, the cell number of the upper-right corner cell is 4, the decimal equivalent for the cell's coordinates, which are A B C = 100.

Because of this one-to-one mapping between cells in K-maps and rows in truth tables, each cell in a K-map corresponds to exactly one minterm and exactly one maxterm, in the same way that a row in a same-size truth table does. For example, cell 4 in Figure 1*a* corresponds to minterm (A . B' . C') and maxterm

[2] For automatic minimization of (large) functions other algorithms such as Quine-McCluskey or iterative consensus are utilized.

[3] From now on, by a K-map, we mean a 2-, 3- or 4-variable K-map, unless otherwise specified.

(A' + B + C) because row 4 in Figure 1*b* corresponds to this minterm and maxterm, as we saw in Chapter 3.

Both the K-map and truth table in Figure 1 are blank, as output information has not been entered yet. To generate a non-blank K-map equivalent to a non-blank truth table we need to copy all the output-column entries of the truth table into the corresponding cells of a same-size K-map. In the rest of this chapter we will drop the adjectives "blank" and "non-blank", as what is meant should be clear from the context. Figure 2*a* and Figure 2*b* show a truth table and the corresponding K-map, respectively. For example, since row 5 in the truth table has a 0 in the output column, cell 5 in the K-map also receives a 0, and is called a *0-cell* or an *off-set* cell. Similarly, the output in row 3 of the truth table is 1; therefore a 1 is entered in cell 3 of the K-map, and so on. Cell 3 is called a *1-cell* or an *on-set* cell.

Row	A B C	y
0	0 0 0	1
1	0 0 1	0
2	0 1 0	0
3	**0 1 1**	**1**
4	1 0 0	0
5	**1 0 1**	**0**
6	1 1 0	0
7	1 1 1	1

(a)

C \ AB	**00**	**01**	**11**	**10**
0	0 1	2 0	6 0	4 0
1	1 0	3 **1**	7 1	5 **0**

(b)

Figure 2. Same function described by a: (a) 3-variable truth table, (b) 3-variable K-map

A two-variable K-map and a four-variable K-map are shown in Figure 3*a* and Figure 3*b*, respectively. Pay

B \ A	**0**	**1**
0	0	2
1	1	3

(a)

CD \ AB	00	**01**	**11**	10
00	0	4	**12**	8
01	**1**	5	13	**9**
11	3	**7**	**15**	11
10	2	6	**14**	10

(b)

Figure 3. (a) 2-variable K-map, (b) 4-variable K-map

close attention to the order in which input bit patterns appear vertically or horizontally (in a K-map). This order is different from that of conventional binary numbers. For example, consider the bit patterns placed horizontally on the top of the map in Figure 3*b*, which reads: 00, 01, 11 and 10. But traditional 2-bit binary numbers have a different order, namely 00, 01, 10 and 11. The (new) code used in K-maps is called *Gray code* and will be addressed in more detail in Chapter 5. The distinguishing characteristic of this code is that two consecutive bit patterns in Gray code, or technically speaking two consecutive Gray *code words* differ in *exactly* one bit[4] as you see in Figure 3*b* for the 2-bit Gray code: for example, after **01** the following code word is **11**, (and not 10) which differs from the preceding code word in the first bit

[4] But the converse is not necessarily true.

only. A similar relationship exists between every two consecutive Gray code words. This, however, is not the case with the traditional binary system: 10 (decimal 2) immediately follows 01 (decimal 1), but they are different in more than one bit. In addition to consecutive Gray code words (positioned next to each other on K-maps), the beginning and ending Gray code words, such as **0**0 and **1**0 in Figure 3*b*, are also different from each other in only one bit.

Question 1. What is the point in using Gray code in K-maps?

We answer this question on page 5 and after we have developed sufficient background.

Definition: In a K-map we call two cells *logically adjacent* if their coordinates differ in only one bit.

For example cells 7 and 15 in Figure 3*b* are logically adjacent because their coordinates are ABCD = 0111 and ABCD = 1111, respectively, which differ in one bit, A, only.

Definition: We call two minterms logically adjacent if they differ in only one variable (i.e., one variable is inverted in one minterm while the same variable is non-inverted in the other minterm, and the remaining literals are shared by the two minterms), e.g. A . **B'** . C' and A . **B** . C' (in a 3-variable truth table), which only differ in the second variable.

Conclusion: It can be shown that two minterms are logically adjacent if they belong to two logically adjacent cells and vice versa.

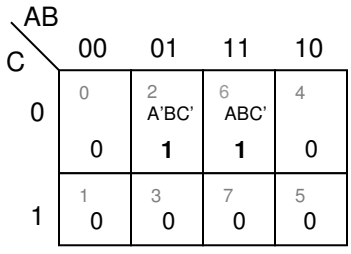
For example, minterms A' . B . C . D and A . B . C . D belong to two logically adjacent cells 7 and 15, respectively (see Figure 3*b*). On the other hand, these two minterms are logically adjacent because they are different from each other in one variable only: A is inverted in the first minterm but not inverted in the second one.

Example 1. Use switching algebra to simplify $Y = \sum_{A, B, C} (2, 6)$.

The canonical SOP of this function is Y = A' . B . C' + A . B . C', and its truth table and K-map are shown in Figure 4*a* and Figure 4*b*, respectively.

Row	A B C	Y
0	0 0 0	0
1	0 0 1	0
2	0 1 0	**1**
3	0 1 1	0
4	1 0 0	0
5	1 0 1	0
6	1 1 0	**1**
7	1 1 1	0

(a)



(b)

Figure 4. For Example 1: (a) 3-variable truth table, (b) 3-variable K-map

Apply T10-L (the combining theorem on the left) to the two p-terms:

T10 a . b + a . b' = a (a + b) . (a + b') = a Combining

A' . B . C' + A . B . C' = B . C'

Therefore, Y = B. C'

The resulting expression needs 1 two-input AND gate only[5], while the original expression needs 2 three-input AND gates and 1 two-input OR gate: a significant saving!

[5] Inverters do not count in the total cost, unless otherwise specified.

Minterms[6] A' . B . C' and A . B . C' are logically adjacent because they differ in only one variable. Therefore, we may come to the conclusion that two logically-adjacent minterms, hence two logically-adjacent 1-cells, can be combined resulting in a simpler p-term. In general, while two *n*-bit adjacent minterms need 2 *n*-input AND gates and 1 two-input OR gate to get realized, the combined p-term needs only one (*n*-1) input AND gate to get realized.

Conclusion: In order to minimize a logic function, we need to identify all the logically adjacent on-set minterms or logically-adjacent 1-cells of the function in hand.

Therefore, our *intermediate goal* is to identify all the logically adjacent 1-cells of the function under consideration.

Definition: In a 2-, 3- or 4-variable K-map, we call two cells *physically adjacent* if they have (at least) one common side.

For example, cells 7 and 15 in the K-map of Figure 5*a* are physically adjacent, as they have one side in common. But cells 3 and 6, for example, are not adjacent, although they have one point in common. Additionally, from this definition's point of view, we need to assume that the top and bottom sides of every K-map are the same line segment, and the right and left sides of every K-map are the same line segment as well. According to this convention cells 1 and 9 in Figure 5*a* do have a common side, so do cells 12 and 14, and so on. As another example consider the physically adjacent cells 4 and 5 in Figure 5*b* which have *two* sides in common. Now we should be able to answer question 1.

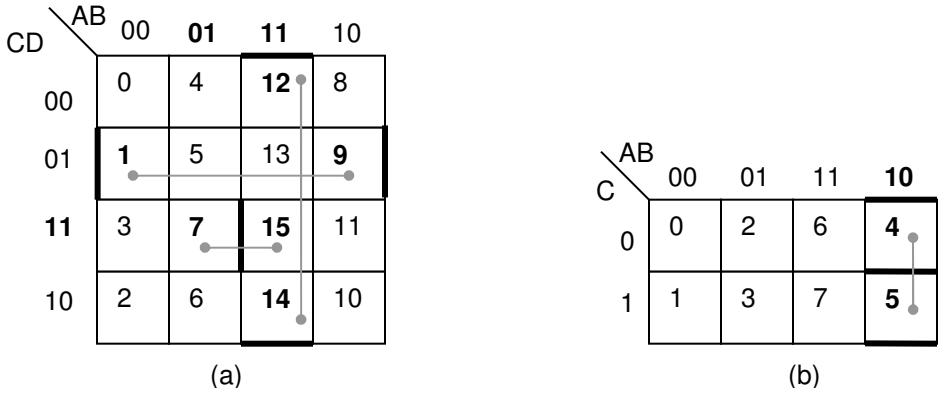


Figure 5. Some physically adjacent cells

Answer 1. By using Gray code in K-maps, physically-adjacent cells become logically adjacent as well, and vice versa.

Question 2. Why is it important to make physically-adjacent cells logically adjacent as well (and vice versa)?

Answer 2. Remember that our intermediate goal is to identify all the logically-adjacent 1-cells. On the other hand, physically-adjacent cells can be identified by human beings at a glance. Therefore, when we know that physically-adjacent 1-cells are logically adjacent as well, and vice versa, then we would easily be able to identify all the logically-adjacent 1-cells at a glance, which is our intermediate goal. ◊

We are now ready to see how the K-map shown in Figure 4*b*, as an example, can help us simplify the corresponding function $Y = \sum(2, 6)$. This K-map is shown again in Figure 6. In this K-map cells 2 and 6 are identified as two physically-adjacent, hence logically-adjacent 1-cells on the spot. But we know that two logically adjacent 1-cells correspond to two logically adjacent minterms; additionally, two logically

[6] By "minterm" in an expression we mean an on-set minterm, which is clear from the context.

adjacent minterms can be combined to reach a smaller p-term as shown in Example 1. Here is a simple procedure to merge these two (or similar) minterms without any explicit need for switching algebra:

The two adjacent minterms in this example are A' . B . C' and A . B . C' as shown in Figure 6. Variable A is inverted in one minterm and non-inverted in the other one. Consider the other two variables, B and C: B is inverted in neither of the two minterms, and C is inverted in both of the minterms. To merge these two minterms the rule is: drop A (which appears as two different literals in the two minterms) and keep B and C' because each of them is either inverted or non-inverted in *both* minterms. Therefore, the p-term resulting from combining the two minterms would be B . C'.

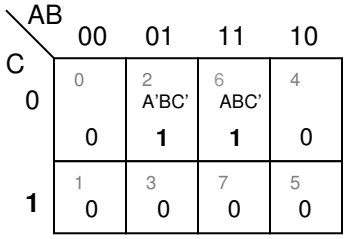


Figure 6. K-map-based minimization: the basic concept

The following is a summary of how to combine two adjacent minterms:

Two adjacent minterms can always be combined to produce one single *p-term* with *one variable fewer* than the total variables in each minterm. To combine them, drop the only variable which appears as two different literals in the two minterms and keep the remaining literals. ◊

To keep K-maps as readable as possible, we normally drop 0s from K-maps; in other words the cells with no entries are considered the off-set cells of the function[7].

Example 2. Use a K-map to minimize $Y = \sum_{A, B, C} (2, 3)$.

The canonical SOP of this function is Y = A' . B . C' + A' . B . C, and its K-map with two 1-cells and the corresponding minterms are illustrated in Figure 7*a*. These cells are physically, hence logically adjacent.

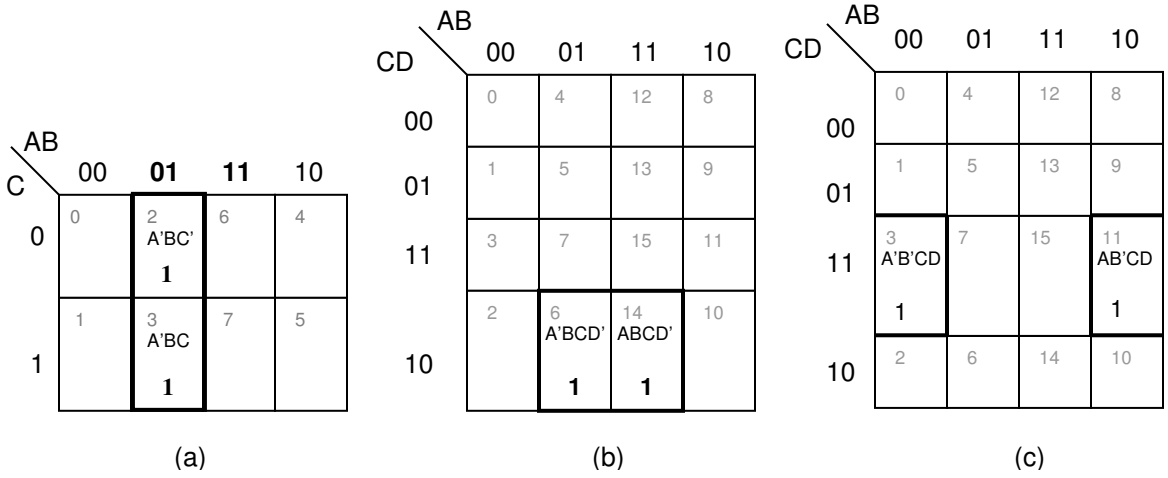


Figure 7. K-map for (a) Example 2, (b) Example 3, (c) Example 4

[7] After the concept of *incompletely specified functions* is introduced later in this chapter, we will see that a K-map may be partitioned into three different types of entries instead of two.

To combine them and obtain the minimized p-term, variable C is dropped because it appears as two different (inverted and non-inverted) literals in the two minterms; but variables A and B appear as A' and B, respectively, in both minterms. Therefore, literals A' and B are kept as they are, resulting in Y = A' . B. This minimized minterm represents a larger cell made up of the two original 1-cells as illustrated in Figure 7*a*. On the other hand, for both 1-cells the value of variable A is 0 and the value of variable B is 1, as the top coordinates in Figure 7*a* show. Therefore, the coordinates of the resulting (larger) cell would be A B = 01, i.e., the ordered values of those variables that do not change within the coordinates of the participating 1-cells.

A slightly different but simpler approach may be adopted to combine these (or similar) 1-cells and obtain the minimized p-term. Consider the coordinates of these two cells: ABC = 010 for cell 2, and ABC = 011 for cell 3. Variables A and B each have a fixed value (A = 0 and B = 1) in the coordinates of both cells; however, this is not the case with C, which is 0 in the coordinates of cell 2 but 1 in the coordinates of cell 3. To combine cell 2 and cell 3, and obtain the minimized p-term we need to drop variable C and keep the other two, A and B. Since the value of A is 0 in the coordinates of both cells, A is inverted and then participates in the minimized p-term. However, the value of B is 1 in the coordinates of both cells, hence B appears as a non-inverted variable in the final p-term, so Y = A' . B.

The second procedure (to combine two adjacent 1-cells) explained above may be summarized as follows:

Rule 1. Obtain the right variables: determine the only variable that is not fixed in the coordinates of the two 1-cells to be combined. Discard this variable and keep the rest.

Rule 2. Obtain the right primes (or negations): If the fixed value of a variable (which was kept according to Rule 1) is 1, that variable will participate in the minimized p-term as a *non-inverted* variable; otherwise, the variable will be *inverted*. The resulting p-term represents a larger rectangular cell comprised of the two original 1-cells. ◊

Example 3. Use a K-map to minimize $Y = \sum_{A, B, C, D} (6, 14)$.

These two 1-cells and their minterms are shown in Figure 7*b*. Again, it is easy to see that these two 1-cells are physically, hence logically adjacent. To combine them variable A is dropped according to Rule 1, because A is 0 in the coordinates of cell 6, but it is 1 in the coordinates of cell 14. On the other hand, variables B, C and D are 1, 1 and 0, respectively, in the coordinates of both cells resulting in Y = B . C . D', according to Rule 2. This p-term represents a larger cell comprised of the two original 1-cells as shown in Figure 7*b*.

Example 4. Use a K-map to minimize $Y = \sum_{A, B, C, D} (3, 11)$.

These two 1-cells and their minterms are illustrated in Figure 7*c*. According to the assumption made earlier in this chapter regarding the right and left sides of a K-map, these two 1-cells are physically, hence logically adjacent and therefore can be combined. To combine them variable A is dropped according to Rule 1, because A is 0 in the coordinates of cell 3, but it is 1 in the coordinates of cell 11. On the other hand, variables B, C and D are 0, 1 and 1, respectively, in the coordinates of both cells, resulting in Y = B' . C . D, according to Rule 2. This p-term represents a cell twice as large as the two original 1-cells as illustrated in Figure 7*c*.

Repetitive Combining: An Extension to Single-cell Combining

We have been able to combine two adjacent 1-cells to reach a larger cell, hence a smaller p-term. But this is only the first major step towards logic minimization; more work has to be carried out to reach the goal.

Example 5. Use switching algebra to minimize $Y = \sum_{A, B, C, D} (1, 3, 5, 7)$.

The canonical SOP of Y is as follows:

Y = **A'** . **B'** . C' . **D** + **A'** . **B'** . C . **D** + **A'** . **B** . C' . **D** + **A'** . **B** . C . **D**

Apply T10-L to the first two and the second two p-terms:

$Y = \mathbf{A'} \cdot \mathbf{B'} \cdot \mathbf{D} + \mathbf{A'} \cdot \mathbf{B} \cdot \mathbf{D}$

Let's see what has happened so far in the K-map domain. 1-cells 1 and 5 have been combined with 1-cells 3 and 7, respectively, as shown in Figure 8*a*. This combining is of course not new to us.

Apply T10-L again to the resulting two p-terms:

$Y = \mathbf{A'} \cdot B' \cdot \mathbf{D} + \mathbf{A'} \cdot B \cdot \mathbf{D} = \mathbf{A' \cdot D}$ (1)

Y is eventually minimized to A' . D. In (1) combining was carried out in the algebraic domain, but we do not know its counterpart in the K-map domain yet. This is the issue to be addressed in this section, where the concepts developed so far (to combine two single 1-cells) are extended to combine as many legitimate 1-cells as possible, as shown in Figure 8*b* for Example 5: the two pairs of 1-cells obtained in Figure 8*a* are combined *again* to reach an even larger cell comprised of cells 1, 3, 5 and 7.

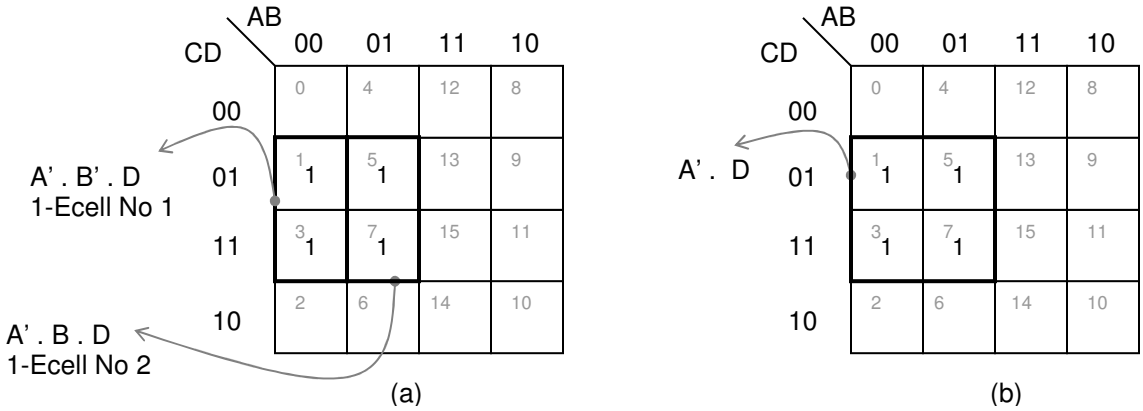


Figure 8. K-maps for Example 5: (a) first stage of combining, (b) second stage of combining

We learned that when two adjacent 1-cells are combined, then: 1) in the algebraic domain the two minterms corresponding to these two 1-cells are replaced with one smaller p-term that has one literal fewer than either of the original minterms, and 2) in the K-map domain the two 1-cells are combined and a rectangular cell twice as large is created. We call the resulting cell an *extended cell*, or *Ecell* for short. More specifically, since this Ecell is built up of only 1-cell, it is called an *on-set Ecell* or *1-Ecell*, which is represented by the newly generated p-term and vice versa. A 1-Ecell has its own coordinates, which are the ordered values of those variables that do not change within the coordinates of the participating 1-cells. For example, cells 1 and 3 shown in Figure 8*a* are combined and 1-Ecell No 1 (the union of 1-cells 1 and 3) is created as highlighted in this figure. In other words, minterms A'. B'. C'. D and A' . B' . C . D, which correspond to cells 1 and 3, respectively, are combined to produce a smaller p-term, namely A'. B'. D, representing 1-Ecell No 1 shown in Figure 8*a*. In the coordinates of both cell 1 and cell 3 the values of A, B and D are 0, 0, and 1, respectively, but variable C has different values in these coordinates. So, by definition, the coordinates of the resulting E-cell (E-cell No 1 in Figure 8*a*) would be A B D = 001. In summary, as two minterms are combined, the resulting p-term loses one variable (shrinks), and the resulting Ecell doubles in size (grows).

An Ecell comprised of 0-cells only is called a *0-Ecell* or an *off-set Ecell*. We may leave prefix 1 and/or prefix 0 out if what is intended is clear from the context.

1-Ecell Combining

Similar to two adjacent 1-cells, two adjacent 1-Ecells (defined below) can always be combined as well, resulting in a larger 1-Ecell. This in fact is a *recursive* definition for 1-Ecells. Additionally and without the loss of generality, a single 1-cell may also be called a (single-cell) 1-Ecell. Therefore, and from now

on, 1-Ecells can take different sizes. We already know how to generate two-cell 1-Ecells. The way that larger 1-Ecells are obtained is elaborated on here.

Two same-size Ecells are *physically adjacent* if they have (at least) one *same-size* side in common, such as the two 1-Ecells shown in Figure 8*a*. (Remember that the top and bottom sides of every K-map are considered the same line segment, and the two right and left sides of every K-map are considered the same line segment as well.) Different-size Ecells can *never* be adjacent. On the other hand, two same-size Ecells are logically adjacent if within the coordinates of these Ecells the value of *only one* variable changes (so, the remaining variables each stay at a fixed value). For example, the coordinates of 1-Ecell No 1 and 1-Ecell No 2 in Figure 8*a* are A B D = 001 and A B D = 011, respectively, which are only different in the value of variable B; therefore these two physically adjacent 1-Ecells are logically adjacent as well. *It can be shown that two physically adjacent E-cells are also logically adjacent and vice versa.*

Two p-terms are logically adjacent if they are made up of the same subset of variables out of which exactly one variable is inverted in one of the p-terms and non-inverted in the other one. For example, p-terms A' . B' . D and A' . B . D are logically adjacent, but p-terms A' . B . C and A' . B . D are not. *It can be shown that two logically adjacent p-terms represent two logically adjacent 1-Ecells and vice versa.*

Two adjacent 1-Ecells can be combined resulting in another (rectangular) 1-Ecell twice as large. The resulting 1-Ecell is represented by a p-term comprised of all the literals *shared* by the adjacent p-terms of the two original 1-Ecells. Therefore, the resulting p-term will have one literal fewer than each merging p-term has. This combining procedure between newly-generated adjacent 1-Ecells may continue until no combining is possible anymore. Remember that the more 1-Ecells are combined, the larger the resulting 1-Ecell, hence the smaller (less expensive) the resulting p-term is.

Considering that in every combining stage the resulting (rectangular) 1-Ecell is twice as large as each merging 1-Ecell/1-cell, the total number of participating 1-cells in a 1-Ecell has to be a power of 2. In other words, *any rectangle with 2^k 1-cells is a 1-Ecell, where k is an integer*. For k = 0 the number of participating 1-cells in the 1-Ecell becomes 1, signifying a single-cell 1-Ecell.

Definition: An on-set Ecell/cell is called a *prime implicant* if it cannot grow anymore. In the algebraic domain the p-term representing a prime implicant is also called a prime implicant.

Therefore, a prime implicant has the fewest number of literals (i.e., it needs the smallest AND gate to get realized) comparing with the 1-Ecells/1-cells comprising that prime implicant.

For example, 1-Ecell A . C' shown in Figure 9*a* (which is made up of 1-cells 4 and 6) is a prime implicant, and so is 1-cell 2, A' . B . C', in Figure 9*b*. First consider the 1-Ecell in Figure 9*a*. This Ecell (similar to any other Ecell) may grow vertically, horizontally or in both directions. Its vertical growth will end up with the E-cell comprised of cells 4, 5, 6, and 7. But cell 7 is a 0-cell, hence cannot participate in a 1-Ecell. Similarly, the horizontal growth requires a 1 in cell 0, but this is a 0-cell. Therefore, both vertical growth and horizontal growth are ruled out. In a similar way it can be shown that the single-cell 1-Ecell shown in Figure 9*b* cannot grow either.

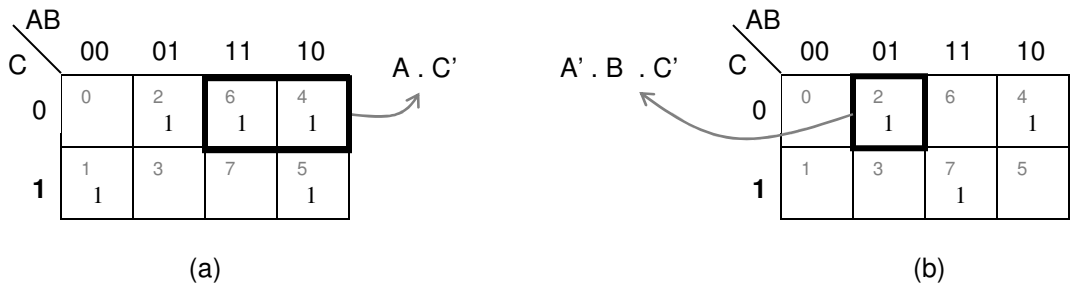


Figure 9. Prime implicant examples: (a) a two-cell prime implicant, (b) a one-cell prime implicant

Conclusion: All the p-terms in a minimal SOP must be prime implicants.

The above conclusion is the basis for Phase 1 of logic minimization described below:

Phase 1 of logic minimization using K-maps: Obtain all the prime implicants of the function in hand.

In the rest of this section we take a close look at some examples to thoroughly understand how to use a K-map to obtain the prime implicant of a single-prime-implicant function. We show how to determine all the prime implicants of a multiple-prime-implicant function (Phase 1 of logic minimization) in the following section and then how to reach a minimal SOP (Phase 2 of logic minimization) later in this chapter.

Example 6. Obtain the prime implicant of $Y = \sum_{A, B, C, D} (8, 9, 10, 11, 12, 13, 14, 15)$.

The K-map of this function is shown in Figure 10*a*. We use the repetitive combining procedure described above to merge as many 1-cells as possible to reach the prime implicant of this function. As the first step we consider the following adjacent-cell pairs:

(12,13) , (8, 9) , (14, 15) , (11,10)

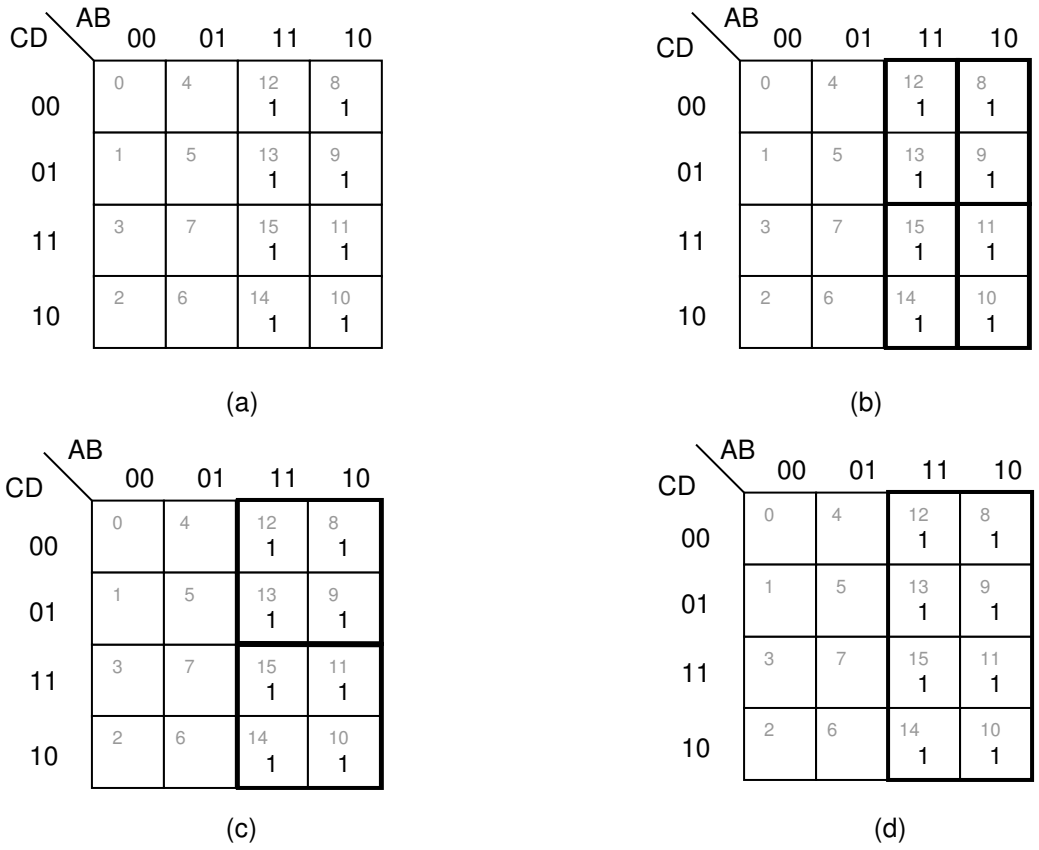


Figure 10. K-maps for Example 6

Notice that we do have other alternatives for cell grouping; for example we could group them horizontally as well. However, the final result would be the same.

The selected pairs have been combined in Figure 10*b* to produce four 1-Ecells as follows:

To combine cells 12 and 13 variable D is dropped resulting in p-term A . B . C', which represents the upper left Ecell.

To combine cells 8 and 9 variable D is dropped resulting in p-term A . B' . C', which represents the upper right Ecell.

To combine cells 14 and 15 variable D is dropped resulting in p-term A . B . C, which represents the lower left Ecell.

To combine cells 11 and 10 variable D is dropped resulting in p-term A . B'. C, which represents the lower right Ecell.

Now the four Ecells in Figure 10*b* may be partitioned into two pairs each comprised of two adjacent Ecells. The two adjacent E-cells in each pair are then combined resulting in two four-cell Ecells in total as shown in Figure 10*c* . The combining procedure is as follows:

To combine the upper two Ecells in Figure 10*b* variable B is dropped resulting in p-term A . C', which represents the upper 1-Ecell in Figure 10*c*.

To combine the lower two Ecells in Figure 10*b* variable B is dropped resulting in p-term A . C, which represents the lower 1-Ecell in Figure 10*c*.

We could group the E-cells in Figure 10*b* vertically as well. However, the final result would be the same.

And finally since the last two Ecells shown in Figure 10*c* are adjacent, they can be combined as well. To combine them variable C is dropped resulting in prime implicant A, which represents the eight-cell 1-Ecell shown in Figure 10*d*. Therefore,

$$Y = A . B'. C'. D' + A . B' . C' . D + A . B'. C . D' + A . B' . C . D + A . B . C' . D' + A . B . C'. D + A . B . C . D' + A . B . C . D = A \lozenge$$

We do not have to perform cell or Ecell grouping one at a time. As soon as we identify a prime implicant in the K-map, we can obtain the corresponding p-term using the following shortcut, which is an extension to Rules 1 and 2. Remember that a prime implicant is any rectangle with 2^k 1-cells that cannot grow anymore, where k is an integer

Rule 3. Obtain the right variables: Check the prime implicant under consideration in the K-map and determine the variables each with a fixed value in the coordinates of all the participating 1-cells. Keep these variables and discard the rest.

Rule 4. Obtain the right primes (negations): If the fixed value of a variable (which was kept according to Rule 3) is 1, then that variable will participate in the p-term as a *non-inverted* variable, otherwise the variable will be *inverted*.

Example 7. Use Rules 3 and 4 to obtain the prime implicant of $Y(A, B, C, D) = \sum(5, 7, 13, 15)$.

The 4 on-set cells of this function comprise 1 prime implicant because the number of participating 1-cells is 2^2 (a power of 2), and additionally the 1-cells form a rectangle, as shown in Figure 11.

As depicted in the top coordinates of the K-map, variable B has the same value, namely 1, for all four cells in this prime implicant. But this is not the case with A, as it is 1 and 0 for cell 13 and cell 5, respectively. Also, as shown in the left coordinates of the K-map, variable D has the same value, namely 1, for all four participating 1-cells. However, variable C is 0 and 1 for cell 5 and cell 15, respectively. Therefore, this prime implicant consists of two variables B and D, according to Rule 3. We need to find out which variables, if any, have to be inverted. In the coordinates of all of these four cells the logic values of both B and D are high. So, according to Rule 4 none of these two variables will be inverted, resulting in the prime implicant B . D for Y.

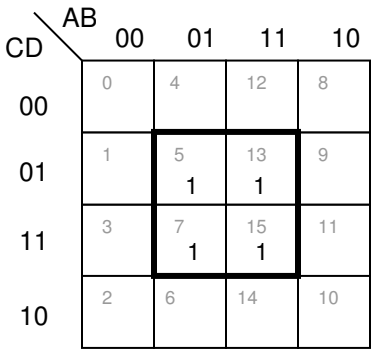


Figure 11. K-map for Example 7

Example 8. In a similar way the prime implicant of each function shown in Figure 12 is obtained as follows:

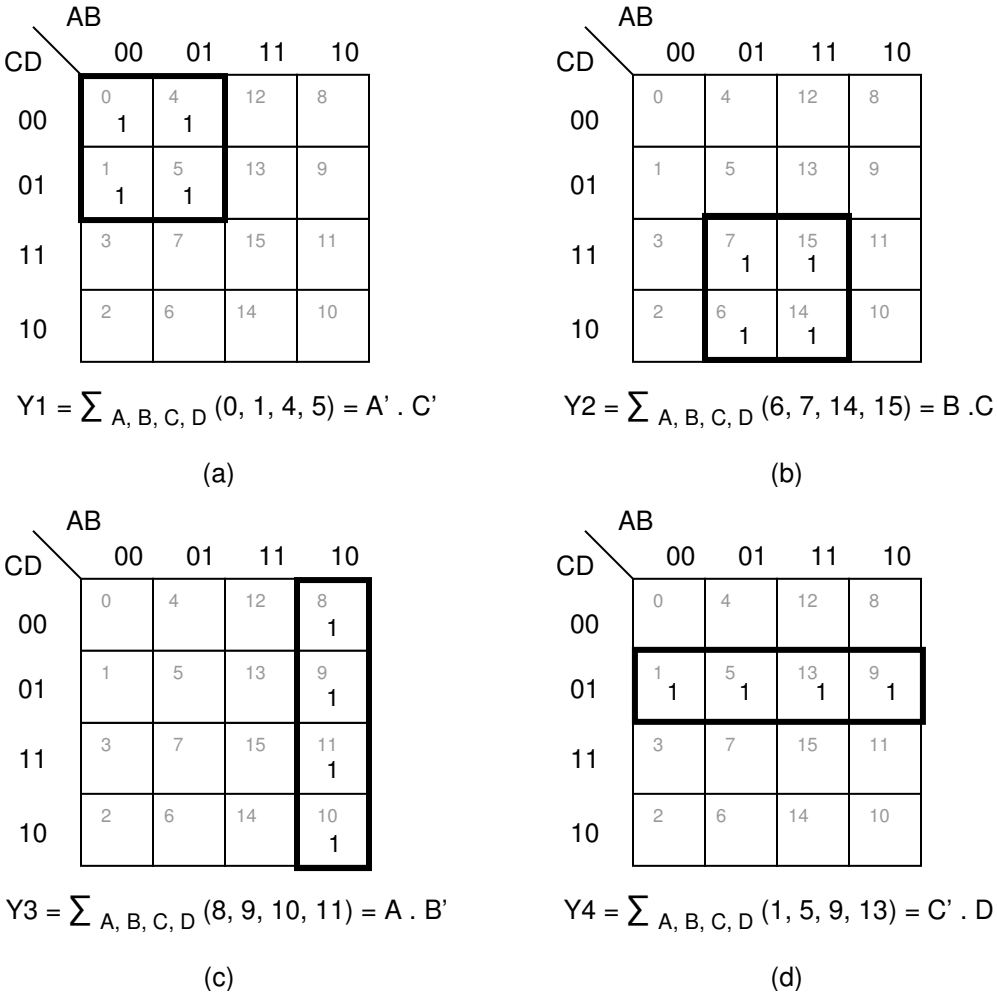


Figure 12. Four single-prime-implicant functions for Example 8

Y1 = A' . C'

Y2 = B . C

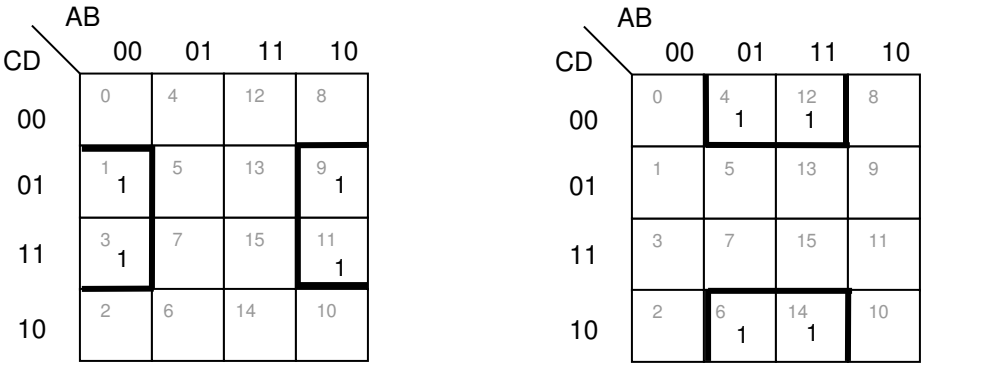
Y3 = A . B'

Y4 = C' . D

Example 9. Use Rules 3 and 4 to obtain the prime implicant of $Y1 = \sum_{A, B, C, D} (1, 3, 9, 11)$.

The four on-set cells of Y1 are shown in Figure 13*a*. Remember that the right and left sides of every K-map are assumed to be the same line segment. Therefore, the four 1-cells shown in Figure 13*a* comprise one rectangle, resulting in the four-cell prime implicant B' . D.

In a similar way the prime implicant of $Y2 = \sum_{A, B, C, D} (4, 6, 12, 14)$ is obtained as B . D'. The K-map of Y2 is shown in Figure 13*b*. Remember that the top and bottom sides of every K-map are assumed to be the same line segment.



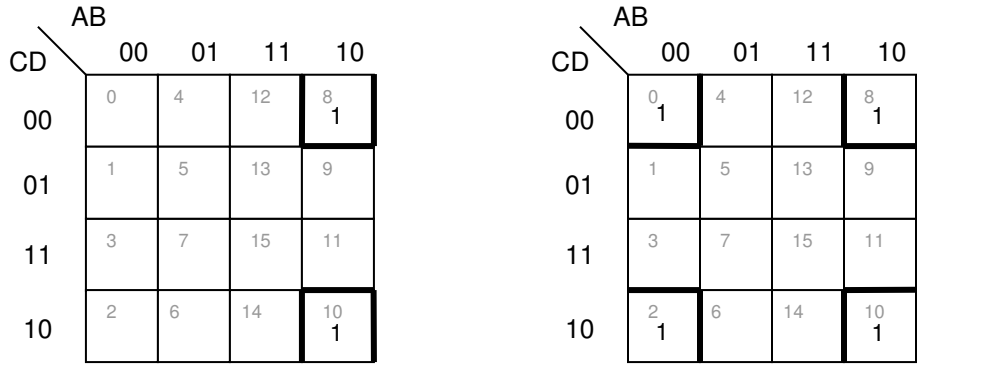
$Y1 = \sum_{A, B, C, D} (1, 3, 9, 11) = B' . D$ (a)

$Y2 = \sum_{A, B, C, D} (4, 6, 12, 14) = B .D'$ (b)

Figure 13. K-maps for Example 9

Example 10. Obtain the prime implicant of $Y1 = \sum_{A, B, C, D} (8, 10)$.

The two on-set cells of Y1 are shown in Figure 14*a*. These two 1-cells can be combined resulting in the two-cell prime implicant A . B' . D'.



$Y1 = \sum_{A, B, C, D} (8, 10) = A . B' . D'$ (a)

$Y2 = \sum_{A, B, C, D} (0, 2, 8, 10) = B' .D'$ (b)

Figure 14. K-map for: (a) Example 10, (b) Example 11

Example 11. Obtain the prime implicant of $Y2 = \sum_{A, B, C, D} (0, 2, 8, 10)$.

The four on-set cells of Y2 are shown in Figure 14*b*. Compare the two K-maps shown in Figure 14*a* and Figure 14*b*. The Ecell in Figure 14*a* has been copied twice into Figure 14*b*. But these two Ecells are adjacent because the right and left sides of a K-map are assumed to be the same line segment. Therefore, the four on-set cells in Figure 14*b* can be combined, resulting in the four-cell prime implicant B' . D'.

Example 12. The on-set cell/Ecell pairs shown in each K-map in Figure 15 cannot be combined because they are not comprised of 2^k 1-cells in a rectangular shape; more specifically they are not adjacent for the following reasons. The two 1-cells in Figure 15*a* do not have a side in common. The same is true for the two 1-cells shown in Figure 15*b*. The two 1-Ecells in Figure 15*c* do not have a side in common. In Figure 15*d* the 1-Ecell on the right does not share one *full* side with its partner. The 1-cell and 1-Ecell in Figure 15*e* do not have the same size. The same is true for the 1-cell/1-Ecell shown in Figure 15*f*.

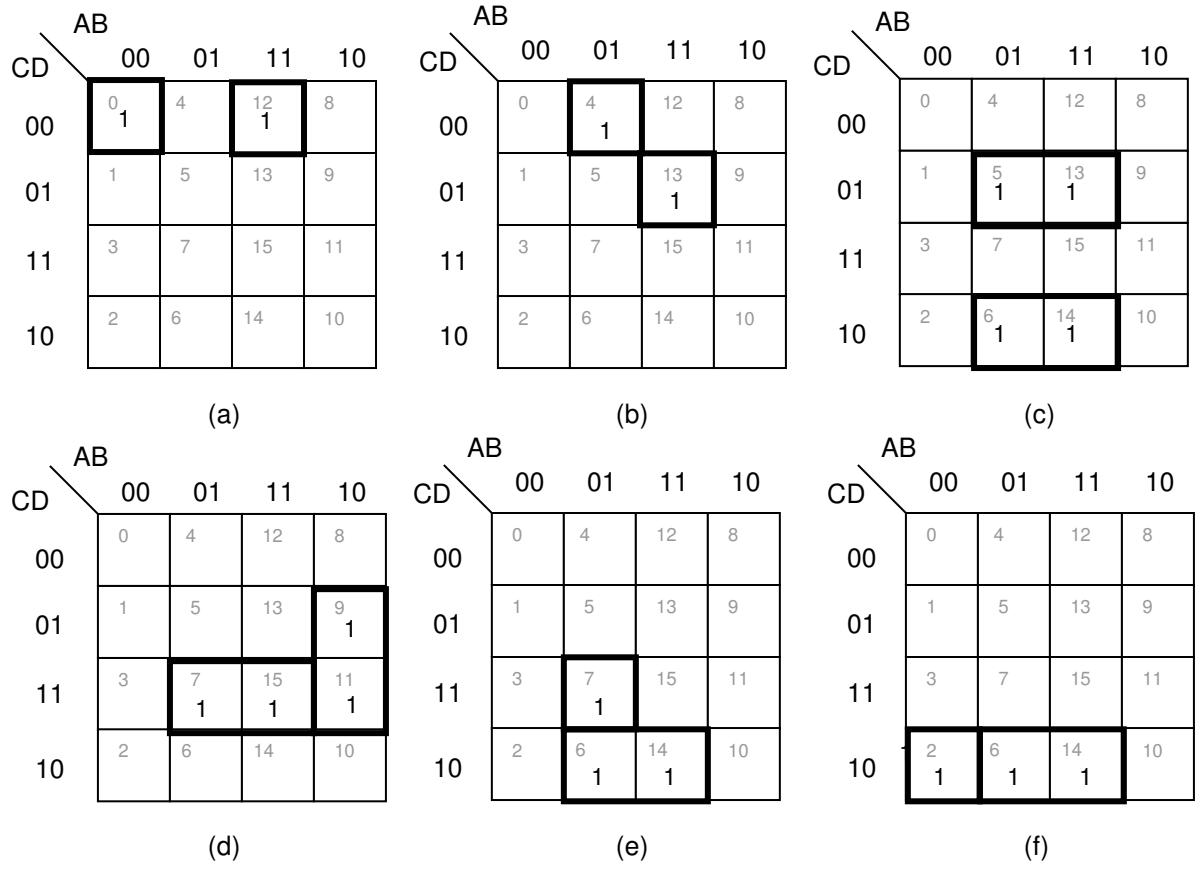


Figure 15. K-maps for Example 12: some non-adjacent cells/Ecells

Complete SOPs versus Minimal SOPs

Definition: A 1-cell is *covered* by a prime implicant if the cell is a member of that prime implicant.

For example, 1-cell 15 in Figure 16 on page 15 is covered by prime implicants 2 and 3, but not by prime implicant 1. From now on prime implicants will be circled to illustrate their intersections more clearly.

In the previous section we considered only single-prime-implicant functions; but a function may have multiple prime implicants. As an example consider $Y = \sum_{A, B, C, D} (7, 9, 13, 15) = A . B' . C' . D + A . B .$

C' . D + A . B . C . D + A' . B . C . D. Figure 16 illustrates a K-map for Y. There are three on-set Ecells circled in this K-map: {**A** . **C'** . **D**, **B** . **C** . **D** and A . B . D}. Since none of them may further grow, these Ecells are prime implicants. On the other hand, 1-cell combining is not possible anymore on this K-map; therefore the above set is the set of all the prime implicants of Y.

Definition: The sum of all the prime implicants of a function is called the *complete SOP*.

It can be shown that the complete SOP of a function has the same truth table by which the function is defined; or the complete SOP is always a correct algebraic expression to represent the corresponding function. However, the complete SOP is not necessarily minimal as we will see shortly.

The complete SOP of the function shown in Figure 16 is as follows:

Y = **A** . **C'** . **D** + **B** . **C** . **D** + A . B . D

Now starting with the canonical SOP of Y, let's use switching algebra to minimize this function. The canonical SOP of Y has been repeated here for ease of reference:

Y = **A** . B' . **C'** . **D** + **A** . B . **C'** . **D** + A . **B** . **C** . **D** + A' . **B** . **C** . **D**

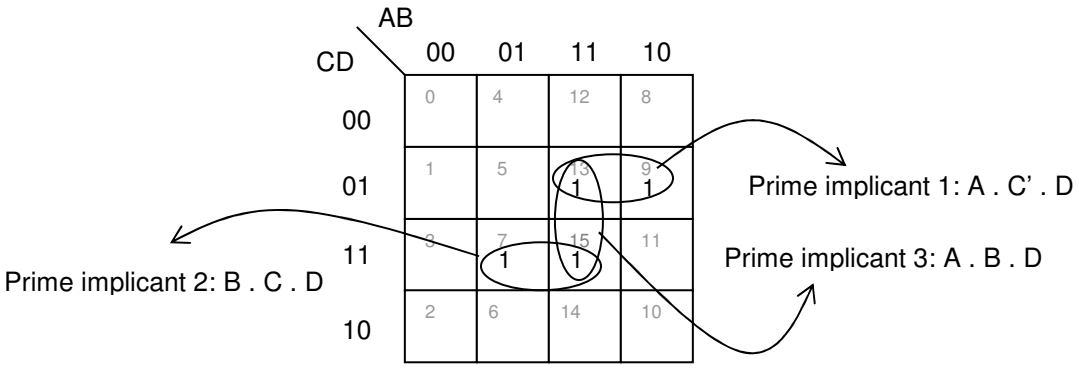


Figure 16. K-map for Y = $\sum_{A, B, C, D}$ (7, 9, 13, 15)

Apply T10-L to the first two minterms and also the last two minterms of Y:

Y = **A** . **C'** . **D** + **B** . **C** . **D**

These two p-terms correspond to prime implicants 1 and 2, respectively, shown in Figure 16. In other words, there are only two (out of three) prime implicants in the above minimized expression for Y. Now the question is: Why is the third prime implicant missing? The answer is: 1-cells 13 and 15 comprising the third prime implicant (A . B . D) are also covered by prime implicant 1 (A . C' . D) and prime implicant 2 (B . C . D), respectively. More specifically, cell 13 belongs to both prime implicants 1 and 3, and cell 15 belongs to both prime implicants 2 and 3 as shown in Figure 16, and therefore prime implicant 3, A . B . D, is redundant and has to be removed from the complete SOP to reach a minimal SOP for Y. And this is what is prescribed by switching algebra as well. Remember that a minimal SOP has the fewest possible prime implicants.

The above example is our first attempt to demonstrate Phase 2 of logic minimization:

Phase 2 of logic minimization using K-maps: Starting with the set of all the prime implicants (of a switching function) determined in Phase 1, obtain a minimum-size subset of that set so that each individual 1-cell will be covered by at least one prime implicant participating in the subset. In case of multiple minimum-size subsets choose the one with the minimum number of literals.

You should notice that the second phase is a *heuristic* procedure, and may not be trivial. In the rest of this section we consider some examples focusing on the second phase. A simple guideline is also developed to let the second phase converge faster.

In Figure 16 we examined a function with the minimal SOP being a subset of the complete SOP. But the complete SOP may also happen to be the minimal SOP, as demonstrated in Example 13.

Definition: A 1-cell is *uncovered* if it is not circled in the K-map.

Example 13. Use a K-map to obtain a minimal SOP for $Y = \sum_{A, B, C, D} (5, 7, 12, 13, 14, 15)$.

Figure 17*b* shows a K-map for Y, which has two prime implicants, namely A . B and B . D, as shown in this figure. So, the complete SOP of this function is as follows:

Y = A . B + B . D

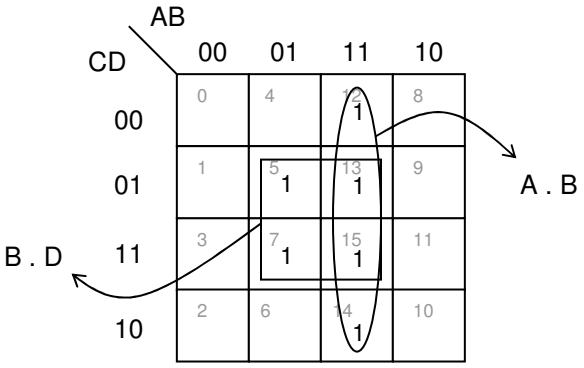


Figure 17. K-map for Example 13

In this example there is no redundant prime implicant in the complete SOP because if either of these two prime implicants is left out, then two 1-cells will be left uncovered. Therefore, in this example the complete SOP is the minimal SOP as well, which in turn means that Y has only one minimal SOP. This however, is not always the case. A switching function may have more than one minimal SOP, as shown in Example 14.

Example 14. Use K-maps to obtain all the minimal SOPs of $Y = \sum_{A, B, C, D} (1, 4, 5, 6, 7, 8, 10, 14)$.

Phase 1: Figure 18*a* shows a K-map for Y, and Figure 18*b* shows all the prime implicants of this function. So the complete SOP of Y is as follows:

Y = A' . B + A' . C' . D + A . B' . D' + A . C . D' + B . C . D'

Phase 2: 1-cell 14 is covered by two prime implicants, namely A . C . D' and B . C . D', as shown in Figure 18*b*. Furthermore, if both of these prime implicants were removed, then the only uncovered 1-cell would be number 14. Therefore, one of these two prime implicants is redundant and has to be left out to reach a minimal SOP. But the question is "which one should be kept?" In general the answer is "the one with the fewest literals". However, in this example both of these candidates have the same size. So, either one can be removed; in other words this function has two minimal SOPs.

In Figure 18*c* prime implicant B . C . D' has been kept resulting in the following minimal SOP:

Y = A' . B + A' . C' . D + A . B' . D' + B . C . D'

In Figure 18*d* prime implicant B . C . D' has been left out resulting in the following minimal SOP:

Y = A' . B + A' . C' . D + A . B' . D' + A . C . D' ◊

The situation could be more complicated than what we examined in Example 14; so that in the beginning it might not be clear what the minimal set of prime implicants is, as demonstrated in the following example.

Example 15. Use K-maps to obtain a minimal SOP for $Y = \sum_{A, B, C, D} (1, 3, 4, 5, 9, 11, 12, 13, 14, 15)$.

Phase 1: Figure 19*a* shows a K-map for Y with a set of five prime implicants, namely {B . C', B' . D, A . B, C' . D, A . D}

Phase 2: What should be clear to us at the first glance is that for this function the complete SOP is not minimal anymore because if prime implicant A . D, as an example, was removed, then each individual 1-cell would still remain covered by one or more prime implicants. However, it is not that simple anymore to identify a minimal SOP by just inspecting the K-map shown in Figure 19*a*. In such cases *distinguished* 1-cells (defined below) are a good staring point to reduce the search space, hence reach a minimal subset of prime implicants faster.

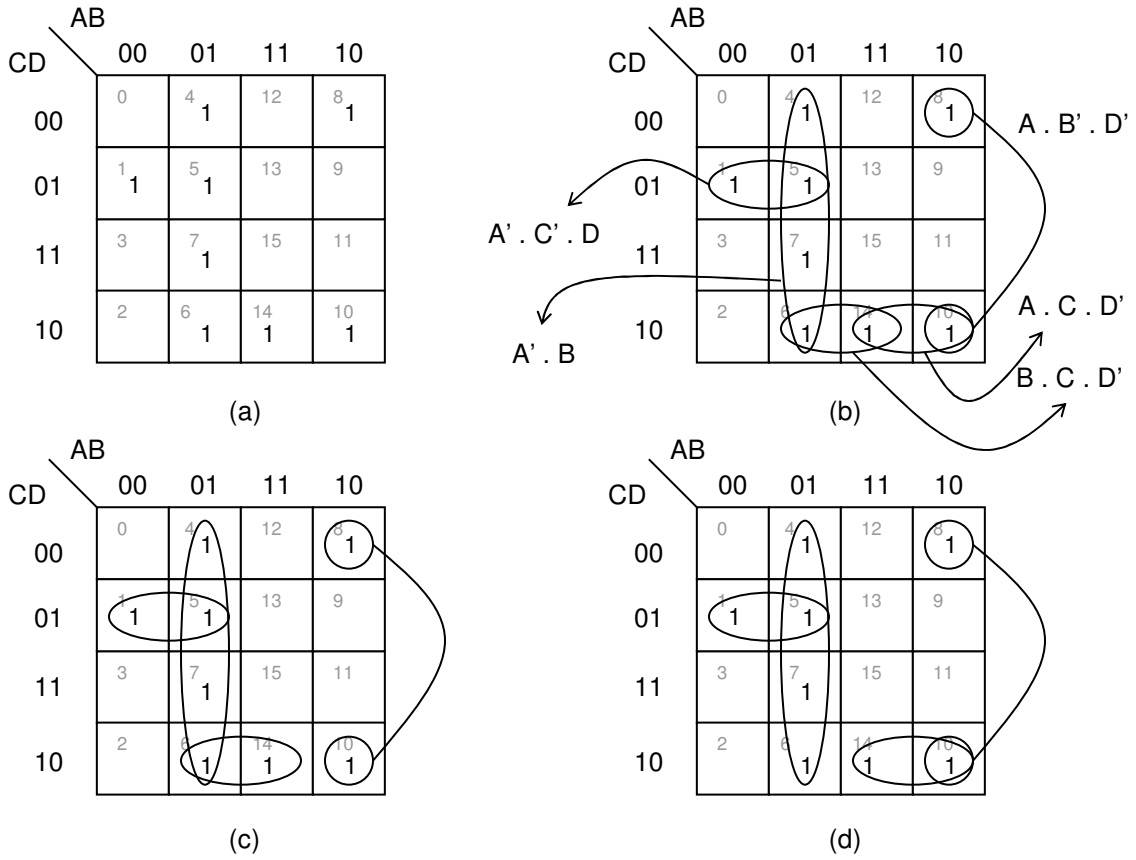


Figure 18. K-maps for Example 14

Definition: A *distinguished* 1-cell of a logic function is a 1-cell that can be covered by *only* one prime implicant.

As soon as all the prime implicants of a function are obtained, it becomes straightforward to identify the distinguished 1-cells, if any: any 1-cell in the K-map that falls in one circle only, is a distinguished 1-cell. Cells 3, 4 and 14, which have been highlighted in Figure 19*a* to Figure 19*c*, are the distinguished 1-cells of Example 15.

Definition: A prime implicant that covers one or more distinguished 1-cells is called an *essential prime implicant*.

Since an essential prime implicant is the *only choice* to cover at least one (distinguished) 1-cell, we arrive at the following conclusion:

Conclusion: All of the essential prime implicants must be included in a minimal SOP.

In Example 15 the essential prime implicants are B . C', B' . D and A . B as shown in Figure 19*b*. The two non-essential prime implicants (i.e., the prime implicants that do not cover any distinguished 1-cells) are shown in Figure 19*c*. In this example, since all the 1-cells of the function are covered by the essential prime implicants, no other prime implicants from Figure 19*c* need to be included in the minimal set of prime implicants, resulting in the following minimal SOP for Y:

Y = B . C' + B' . D + A . B

In this example although the complete SOP is not minimal, the minimal SOP is still unique. ◊

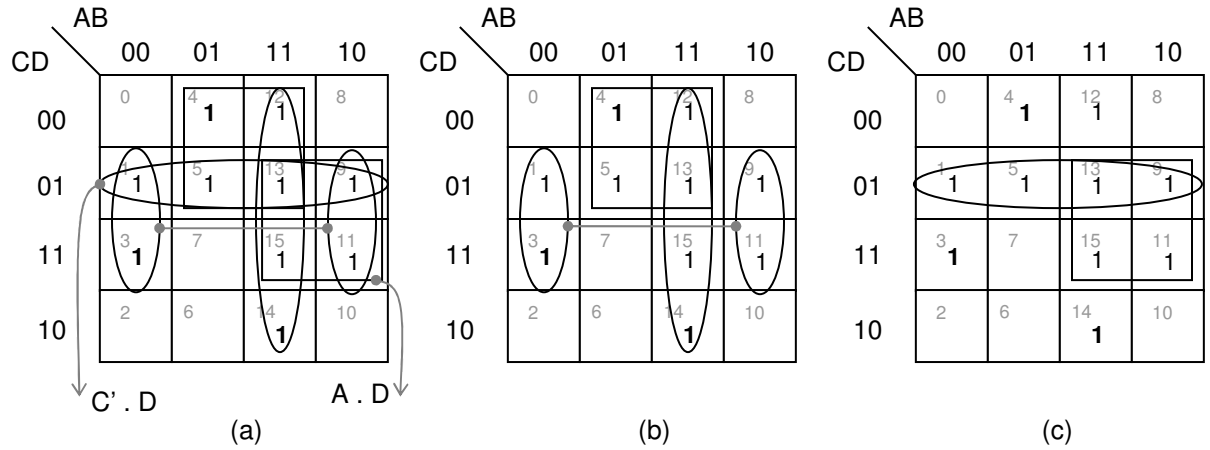


Figure 19. K-maps for Example 15, distinguished 1's have been highlighted: (a) all five prime implicants, (b) essential prime implicants, (c) non-essential prime implicants

In Example 16 in addition to essential prime implicants we also need to include one non-essential prime implicant in the minimal SOP.

Example 16. Use K-maps to obtain a minimal SOP for $Y = \sum_{A, B, C, D} (1, 5, 7, 11, 15)$.

Phase 1: Figure 20*a* shows a K-map for Y with a set of four possible prime implicants, namely {A' . C' . D, A . C . D, A' . B . D, B . C . D}.

Phase 2: This function has two distinguished 1-cells highlighted in all the K-maps of Figure 20. Remember that distinguished 1-cells can easily be identified by inspection. The essential prime implicants and non-essential prime implicants of Y are shown in Figure 20*b* and Figure 20*c*, respectively. As shown in Figure 20*b* cell 7 is not covered by any of the essential prime implicants, while it is covered by two non-essential prime implicants as illustrated in Figure 20*c*. Therefore, we need to include one of these two non-essential prime implicants in the minimal SOP. But again the question is "which one should be chosen?" And the answer is the same as what we had in Example 14: "the one with the fewest literals". In this example again both candidates have the same size, resulting in two different minimal SOPs for Y:

Y = A' . C' . D + A . C . D + A' . B . D

Or

Y = A' . C' . D + A . C . D + B . C . D

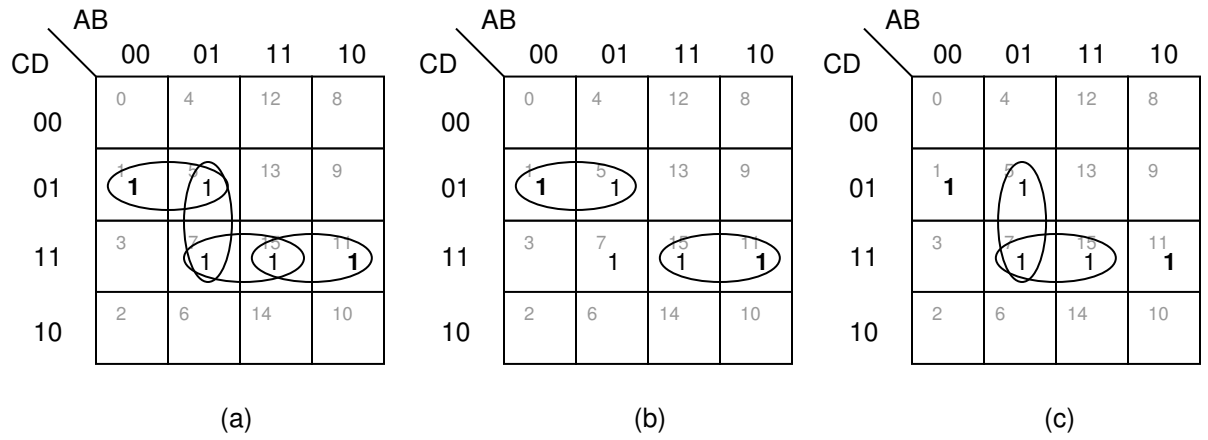


Figure 20. K-maps for Example 16 with highlighted distinguished 1's: (a) all prime implicants, (b) essential prime implicants, (c) non-essential prime implicants

Example 17. Use K-maps to obtain a minimal SOP for $Y = \sum_{A, B, C, D} (1, 3, 5, 6, 7, 9, 11, 13, 14)$.

Phase 1: Figure 21*a* shows a K-map for Y with a set of five possible prime implicants, namely {C' . D, B' . D, B . C . D', A' . D, A' . B . C}

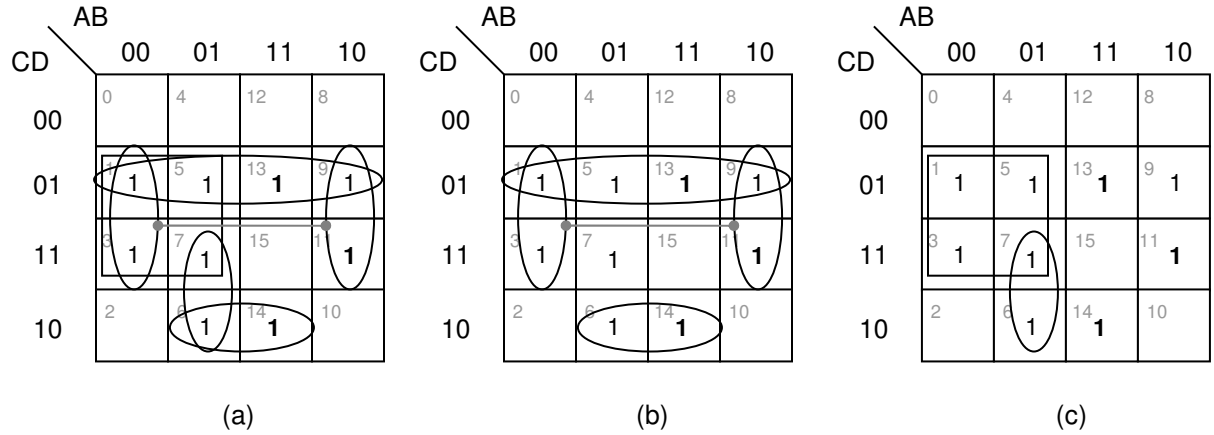


Figure 21. K-maps for Example 17 with highlighted distinguished 1's: (a) all prime implicants, (b) essential prime implicants, (b) non-essential prime implicants

Phase 2: This function has three distinguished 1-cells highlighted in all the K-maps of Figure 21. The essential prime implicants and non-essential prime implicants of Y are shown in Figure 21*b* and Figure 21*c*, respectively. As shown in Figure 21*b* cell 7 is covered by none of the essential prime implicants, while it is covered by two non-essential prime implicants, namely A' . D and A' . B . C, as illustrated in Figure 21*c*. But these two do not incur the same cost; so we must choose A' . D, the one with the fewest number of literals, to reach the only minimal SOP that exists for this function:

$Y = C' . D + B' . D + B . C . D' + A' . D$

Example 18. Single-cell prime implicant: Use a K-map to minimize $Y = \sum_{A, B, C} (3, 4, 6)$

These three 1-cells are shown in Figure 22.

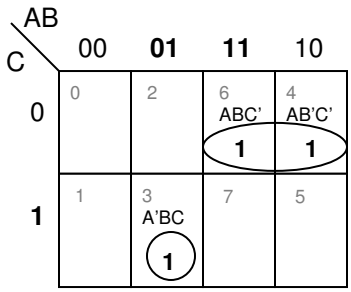


Figure 22. K-map for Example 18: minterms 4 and 6 are merged, but minterm 3 has to be left single

All 1s in this K-map are distinguished; therefore the complete SOP is the minimal SOP as well, i.e.,

$Y(A, B, C) = A' . B . C + A . C'$

The single 1-cell 3 cannot be combined with any 1-cells in the map, resulting in a single-cell prime implicant, $A' . B . C$.

Minimal POS Expressions

In the logic-minimization procedure explained in the previous sections we considered only 1-cells. And that is why we always ended up with SOP expressions. In this section we will group 0-cells instead of 1-cells (but in a similar way), to eventually reach minimal POS expressions. For a given function we may obtain both minimal SOP and minimal POS, and then check to see which expression needs less hardware for possible realization. Remember from Chapter 2 that the number of transistors used in a NAND or a NOR gate is determined by the gate size or number of inputs. Therefore, same-size NAND/NOR gates are considered identical for hardware-cost evaluation purposes. The same is true for same-size AND/OR gates.

We may reuse all the procedures, rules and terms defined in the previous sections but now replacing minterms, p-terms, on-sets, 1s and 0s with maxterms, sum-terms (or s-terms for short), off-sets, 0s and 1s, respectively. Therefore, Rule 3 and Rule 4 are modified as Rule 3' and Rule 4', respectively, shown below. We use these new rules to determine the s-term representing an off-set prime implicant of a logic function mapped to a K-map:

Rule 3'. Obtain the right variables: Check the off-set prime implicant under consideration in the K-map and determine the variables each with a fixed value in the coordinates of all participating 0-cells. Keep these variables and discard the rest.

Rule 4'. Obtain the right primes (negations): If the fixed value of a variable (which was kept according to Rule 3') is 0, then that variable will participate in the s-term as a *non-inverted* variable, otherwise the variable will be *inverted*.

Example 19. Use a K-map to obtain a minimal POS for $Y = \prod_{A, B, C} (0, 1, 3, 5, 7)$.

The two off-set prime implicants of this function are shown in the K-map of Figure 23*a*. First consider the prime implicant on the left for which, variables A and B have fixed values (both are 0s), but this is not the case with C, as it is 0 and 1 for cell 0 and cell 1, respectively. Therefore, this prime implicant consists of two variables, A and B, according to Rule 3'. We need to find out which variables, if any, are inverted. In the coordinates of these two cells the logic values of both A and B are 0s. So, according to Rule 4' none of these two variables will be inverted, resulting in s-term $A + B$.

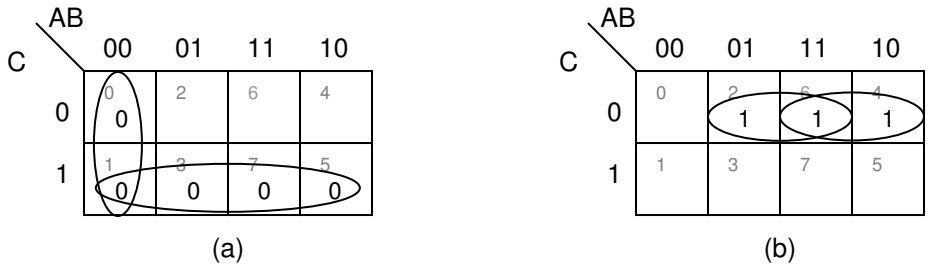


Figure 23. K-maps for Example 19: (a) off-set K-map, (b) on-set K-map

Now consider the second prime implicant comprised of 0-cells 1, 3, 5 and 7 shown in Figure 23*a*. Variable C has a fixed value (which is 1) in all of these 1-cells, but this is not the case with variable A or variable B. Therefore, the second prime implicant consists of variable C only, according to Rule 3'. Then we need to determine whether or not C is inverted. In the coordinates of these four cells the logic value of C is fixed at 1. So, according to Rule 4' C has to be inverted, resulting in the following minimal POS for Y, which needs 1 two-input OR gate and 1 two-input AND gate to get realized.

$Y = (A + B) \cdot C'$

Considering the on-set K-map of Y shown in Figure 23*b*, the minimal SOP of Y is obtained as follows:

$Y = A \cdot C' + B \cdot C'$

This logic expression needs 1 two-input OR gate and 2 two-input AND gates to get realized. Therefore, in this example the minimal POS is more cost-effective than the minimal SOP.

Example 20. Use a K-map to obtain a minimal POS for $Y = \prod_{A, B, C, D} (0, 1, 2, 3, 4, 6, 8, 9, 10, 11)$.

Figure 24 shows all the off-set prime implicants of this function. Using the same procedure explained in Example 19, a minimal POS is obtained for Y as follows:

$Y = B \cdot (A + D)$

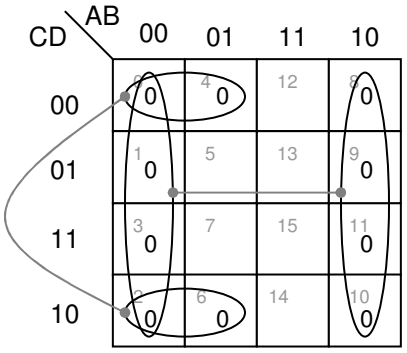


Figure 24. K-map for Example 20

This logic expression needs 1 two-input OR gate and 1 two-input AND, or 2 two-input NOR gates to get realized, as shown in Figure 25*a*.

Figure 25. Bubble-to-bubble logic circuits for Example 20: (a) minimal POS, (b) minimal SOP

A minimal SOP was obtained for the same function in Example 13, and is again shown below for ease of reference:

$Y = A \cdot B + B \cdot D$

This logic expression needs 1 two-input OR gate and 2 two-input AND gates, or 3 two-input NAND gates to get realized as shown in Figure 25*b*. Therefore, the minimal POS for also this function is more cost-effective than the minimal SOP.

Example 21. Use a K-map to obtain a minimal POS for $Y = \prod_{A, B, C, D} (5, 6, 9, 10, 11, 13, 14, 15)$.

All off-set prime implicants of this function are shown in Figure 26*a*, in which, all distinguished 0s have been highlighted. It is clearly seen that all (four) prime implicants of Y are essential. Therefore, in this example (again) the complete SOP is the minimal SOP as well. Using the same procedure explained in Example 19, the minimal POS for Y is obtained as follows:

$Y = (A' + D') \cdot (A' + C') \cdot (B' + C + D') \cdot (B' + C' + D)$

This logic expression needs 2 two-input OR gates, 2 three-input OR gates and 1 four-input AND gate to get realized.

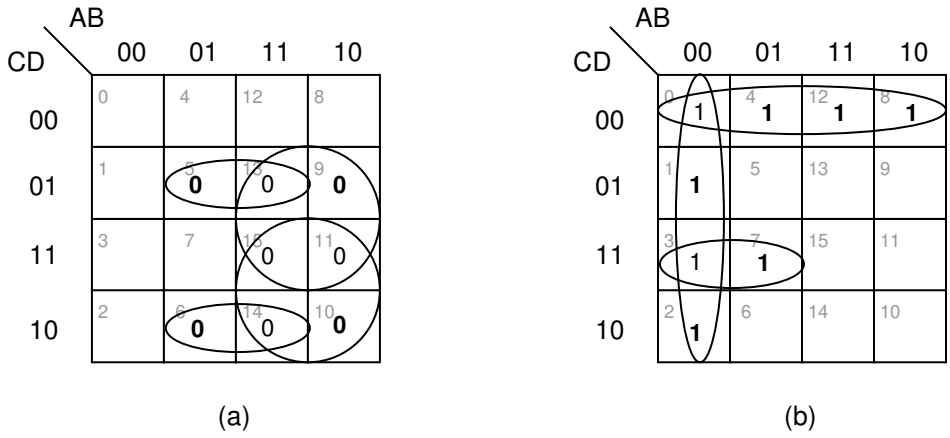


Figure 26. K-maps for Example 21: (a) minimal POS, (b) minimal SOP

On the other hand, Figure 26*b* shows all the on-set prime implicants of the same function, resulting in the following minimal SOP for Y:

$Y = A' \cdot B' + C' \cdot D' + A' \cdot C \cdot D$

This logic expression needs 2 two-input AND gates, 1 three-input AND gate, and 1 three-input OR gate to get realized. Therefore, in this example the minimal SOP is more cost-effective than the minimal POS.

Incompletely Specified Circuits

In Chapter 2 we allowed don't cares to appear in input columns of truth tables to improve the readability of truth tables, and/or to reach more compact truth tables. A don't care may also appear in the output column of a truth table to indicate an impossible input combination as shown in Example 22, or to show this fact that both 0 and 1 outputs (for the corresponding input combination) are consistent with the specifications of the problem in hand, and therefore the designer should not assume a specific output value beforehand for that input combination. A practical example of the second category will be given in Chapter 8.

Example 22. Outstanding junior and senior students are to be identified for the dean's list. They need to have a GPA above 95% and above 90%, respectively, to be chosen. Obtain a truth table for this selection procedure. The five input/output variables are defined as follows:

G95 = 1: GPA above 95%

G90 = 1: GPA above 90%

S = 1: Senior student

J = 1: Junior student

Select = 1: Student is selected

Figure 27 shows a truth table for this example.

Row	G90	G95	J	S	Select	Row	G90	G95	J	S	Select
0	0	0	0	0	0	8	1	0	0	0	0
1	0	0	0	1	0	9	1	0	0	1	1
2	0	0	1	0	0	10	1	0	1	0	0
3	0	0	1	1	x	11	1	0	1	1	x
4	0	1	0	0	x	12	1	1	0	0	0
5	0	1	0	1	x	13	1	1	0	1	1
6	0	1	1	0	x	14	1	1	1	0	1
7	0	1	1	1	x	15	1	1	1	1	x

Figure 27. K-map for Example 22

It is impossible for a student to be senior and junior at the same time. Therefore, any row with a 11 under JS receives a don't care, or x for short, in the output column. It is also impossible for a student to have a GPA above 95%, but not above 90%. Therefore, every row with a 01 under G90 G95 receives a x as well. Each of the rows 9, 13 and 14 gets a 1 in the output column signifying a group of selected students. The remaining rows correspond to non-chosen students, and therefore each one receives a 0 in the output column.

Example 23. Obtain a minimal SOP and a minimal POS for $Y = \Sigma_{A, B, C} (1, 3, 4, 5, 6) + d(0) = \prod (2, 7) \cdot D(0)$. Notice that d(0) and D(0) each represent the don't care set of the function.

For minimization purposes we first replace each don't care entry with a 0 or a 1, whichever results in a more simplified expression. Figure 28*a* shows a minimal SOP in which the don't care cell has been assigned a 1. However, we need to assign a logic 0 to cell 0 in order to reach a minimal POS in this example, as shown in Figure 28*b*.

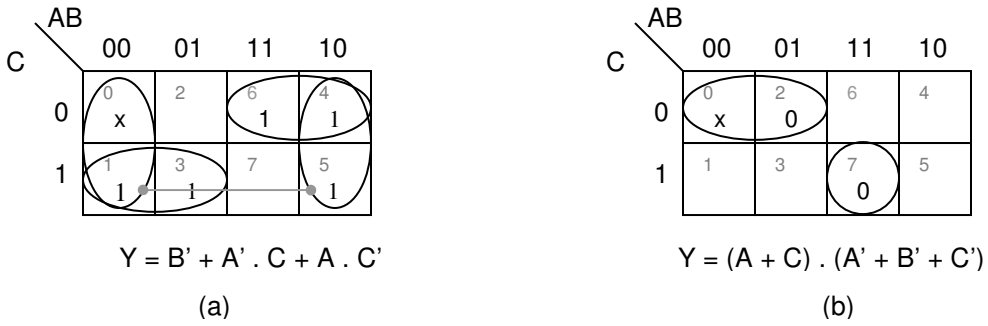


Figure 28. K-map for Example 23: (a) cell 0 receives a 1 to reach minimal SOP, (b) cell 0 receives a 0 to reach minimal POS