4 Karnaugh Maps

In the previous chapter, simplification of expressions for combinatorial logic circuits was studied using Boolean algebra and DeMorgan's theorem. In this chapter, a different graphical based method called Karnaugh maps (or K-maps in short) will be studied to simplify the expressions. But before K-maps can be discussed, the two types of methods for writing logic circuit expressions will be discussed.

4.1 Sum of products

Sum of products (SOP) is a method to express the terms in a logic expression as a sum of products. For example:

$F = ABC + A\overline{B}C$ two product terms ABC and $A\overline{B}C$

$F = AB + A\overline{B} + \overline{A}B$ three product terms AB, $A\overline{B}$ and $\overline{A}B$

The logic circuit diagrams for these expressions are shown in Figure 4.1. It can be seen that each product term is connected using an OR gate.

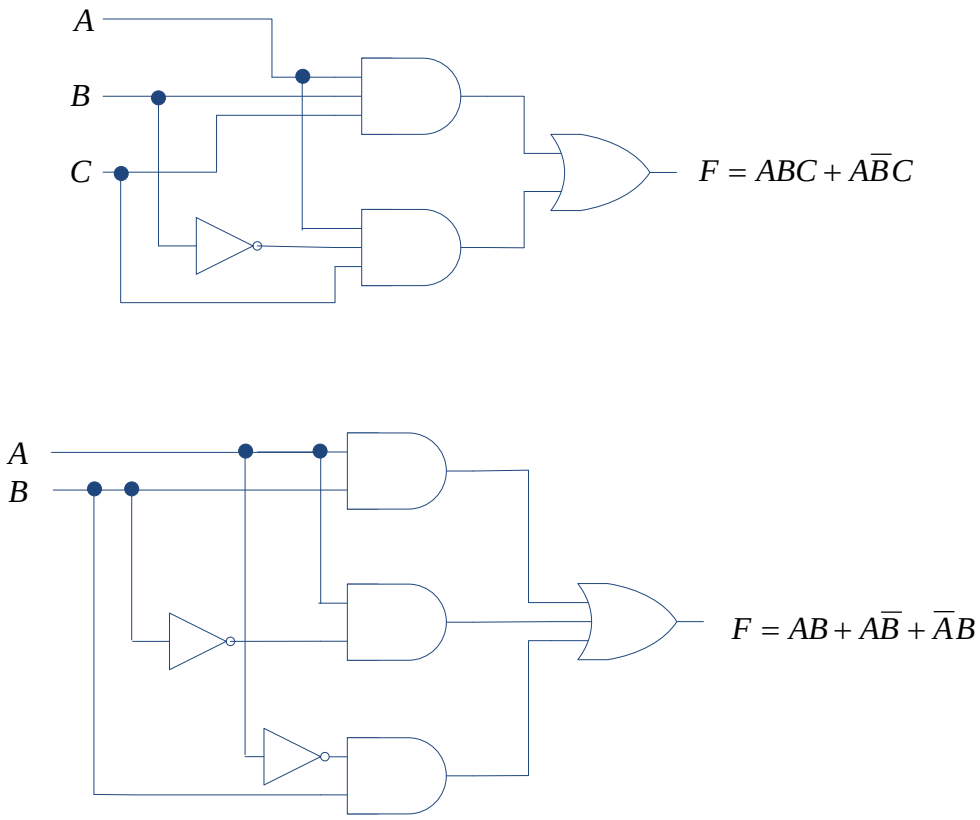


Figure 4.1: SOP logic circuit examples.

Tables 4.1 and 4.2 give the truth tables for these expressions. Each product term results in the output $F = 1$. For example, the expression $F = ABC + A\overline{B}C$ gives output of 1 when A=1, B=1 and C=1 for $F = ABC$ and similarly for $F = A\overline{B}C$, the output is 1 when A=1, $\overline{B} = 1$ (i.e. $B = 0$) and C=1.

Table 4.1: Truth table for $F = ABC + A\overline{B}C$

A	B	C	F	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	$\rightarrow F = A\overline{B}C$
1	1	0	0	
1	1	1	1	$\rightarrow F = ABC$

Table 4.2: Truth table for $F = AB + A\overline{B} + \overline{A}B$

A	B	F	
0	0	0	
0	1	1	$\rightarrow F = \overline{A}B$
1	0	1	$\rightarrow F = A\overline{B}$
1	1	1	$\rightarrow F = AB$

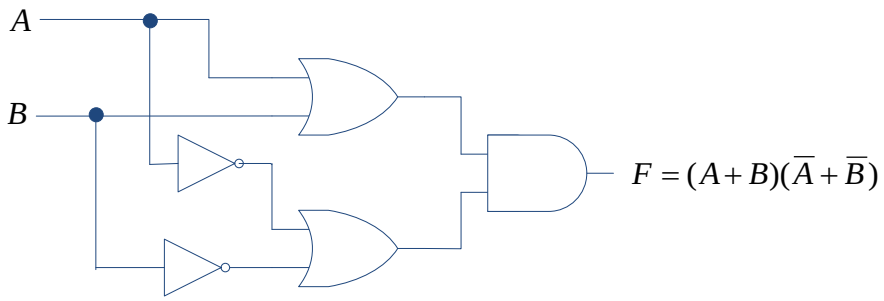
4.2 Product of sums

Products of sums (POS) is another method to express the terms in a logic circuit expression as a product of sums. For example:

$F = (A + B)(\overline{A} + \overline{B})$ two sum terms $(A + B)$ and $(\overline{A} + \overline{B})$

$F = (A + B + C)(A + C)(B + C)$ three sum terms $(A + B + C)$, $(A + C)$ and $(B + C)$

The logic circuit diagrams for these expressions are shown in Figure 4.2. An AND gate connects each of the sum terms.



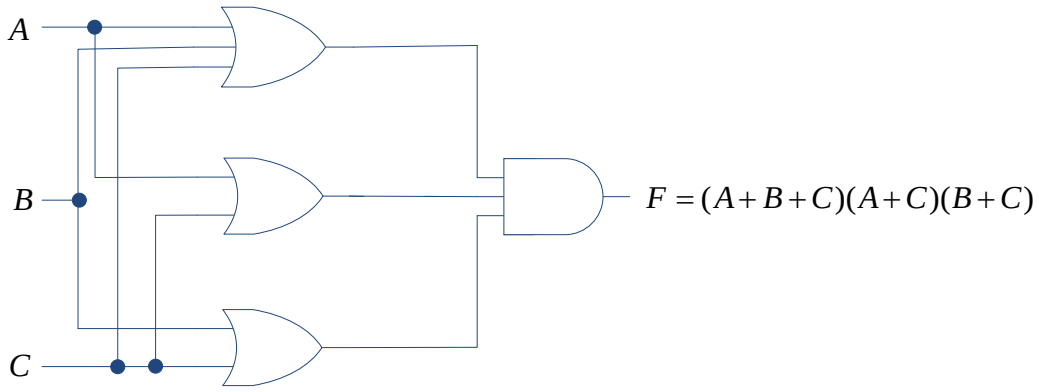


Figure 4.2: POS logic circuit examples.

The truth table for the first POS example, $F = (A+B)(\overline{A}+\overline{B})$ is given in Table 4.3. To understand the table, consider $\overline{F} = \overline{(A+B)(\overline{A}+\overline{B})}$ and using DeMorgan's theorem, we can obtain

$$\overline{F} = \overline{(A+B)(\overline{A}+\overline{B})}$$
$$\overline{F} = \overline{(A+B)} + \overline{(\overline{A}+\overline{B})}$$
$$\overline{F} = \overline{A}\,\overline{B} + AB$$

So, the truth table for POS terms can be easily completed for each term by giving output F=0 with the variables A and B following negative logic (i.e. complemented variable is logic 1 and uncomplemented variable is logic 0).

Table 4.3: Truth table for $F = (A+B)(\overline{A}+\overline{B})$

A	B	F	
0	0	0	$\rightarrow \overline{F} = \overline{A}\,\overline{B}$ or $F = (A+B)$
0	1	1	
1	0	1	
1	1	0	$\rightarrow \overline{F} = AB$ or $F = (\overline{A}+\overline{B})$

Table 4.4 gives the truth table for the second POS example, $F = (A+B+C)(A+C)(B+C)$ . . Following the similar procedure, consider $\overline{F} = \overline{(A+B+C)(A+C)(B+C)}$:

$\overline{F} = \overline{(A+B+C)(A+C)(B+C)}$:

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

$\overline{F} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{C} + \overline{B}\,\overline{C}$

$\overline{F} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{C}(B+\overline{B}) + \overline{B}\,\overline{C}(A+\overline{A})$ since $X + \overline{X} = 1$

$\overline{F} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$

$\overline{F} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C}$ as $\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} = \overline{A}\,\overline{B}\,\overline{C}$

Table 4.4: Truth table for $F = (A+B+C)(A+C)(B+C)$

A	B	C	F	
0	0	0	0	$\rightarrow F = (A+B+C)$
0	0	1	1	
0	1	0	0	$\rightarrow F = (A+C)$
0	1	1	1	
1	0	0	0	$\rightarrow F = (B+C)$
1	0	1	1	
1	1	0	1	
1	1	1	1	

POS expressions are not frequently employed in digital systems but discussed here for the sake of completeness.

4.3 K-maps

As mentioned earlier, K-map is a graphical method that is useful to simplify logic expressions. While the algebraic methods discussed in Chapter 3 can equally be used to simplify the expression, it is often easier to simplify an expression using K-maps when the number of variables is higher.

4.3.1 Two variable K-map

Consider a truth table as in Table 4.5 with two variables A and B. Its corresponding K-map is drawn in Figure 4.3. The K-map can be completed for variable combinations that give F=1 and F=0 as in the figure but it is common practice not to include F=0 in K-maps, so we shall only include combinations that give F=1 after this example.

Table 4.5: Truth table for two variable K-map example

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

		A=0	A=1
		$\overline{A}$	A
B=0	$\overline{B}$	F=0	F=1
B=1	B	F=1	F=1

Figure 4.3: K-map example from truth table in Table 4.5.

To simplify the expression, start by creating a loop for $F = A\overline{B} + AB$ (i.e. for adjacent cells) as shown in Figure 4.4(a). This loop is known as pair loop as it involves looping two 1s. Since $F = A\overline{B} + AB = A(B + \overline{B}) = A$, the looping will result in $F = A$, i.e. the variable in complement and uncomplemented form disappears. The process is repeated until all 1s have been looped (note that loops can overlap). Hence, repeat the looping as shown in Figure 4.4(b) where $\overline{A}B + AB = B(A + \overline{A})$. Since all 1s in the K-map have been looped, further simplification is not possible and the simplified expression is a combination of the two looped terms (each loop gives one term): $F = A + B$.

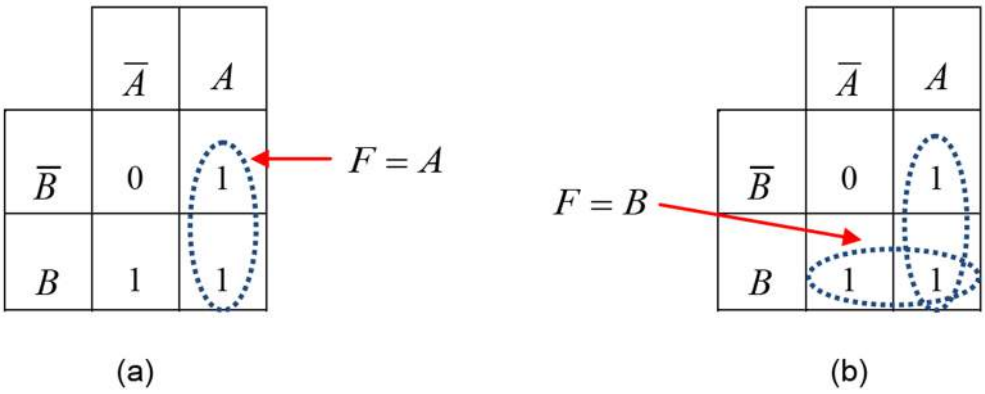


Figure 4.4: Two variable K-map looping: (a) $F = A$, (b) $F = B$. Simplified expression from both loops is $F = A + B$.

Consider solving the example algebraically from the truth table with K-map (each term is a variable combination that gives *F*=1):

$$F = \overline{A}B + A\overline{B} + AB \qquad \text{from Table 4.5 (see section 4.1 on SOP for more details)}$$
$$F = \overline{A}B + A\overline{B} + AB + AB \qquad \text{since } AB = AB + AB$$
$$F = A(\overline{B} + B) + B(\overline{A} + A)$$
$$F = A + B$$

The answer is obviously the same.

4.3.2 Three variable K-map

In addition to pair loops, we can have quad loops (involving four 1s). Consider a three variable logic expression: $F = ABC + A\overline{B}C + \overline{A}\overline{B}C + AB\overline{C} + A\overline{B}\overline{C}$ . . A truth table can be completed with each term ABC, $A\overline{B}C$, $\overline{A}\overline{B}C$, $AB\overline{C}$, $A\overline{B}\overline{C}$ giving output *F*=1 as shown in Table 4.6.

Table 4.6: Truth table for $F = ABC + A\overline{B}C + \overline{A}\,\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C}$.

A	B	C	F	
0	0	0	0	
0	0	1	1	$\rightarrow F = \overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$\rightarrow F = A\overline{B}\,\overline{C}$
1	0	1	1	$\rightarrow F = A\overline{B}C$
1	1	0	1	$\rightarrow F = AB\overline{C}$
1	1	1	1	$\rightarrow F = ABC$

Figure 4.5 gives the completed three variable K-map. Note in particular on the sequence of variables *A* and *B* in the K-map. The sequence (order) follows gray code (00→01→11→10 with $\overline{A}\,\overline{B} \rightarrow \overline{A}B \rightarrow AB \rightarrow A\overline{B}$) where only one bit changes in adjacent cells. Figure 4.6(a) shows the quad loop applied for four adjacent 1s. Variables *B* and *C* are in complemented and uncomplemented forms in the quad loop, so these variables will disappear leaving only variable *A*. For this loop, algebraically,

$$F = ABC + A\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C}$$
$$F = AB(C + \overline{C}) + A\overline{B}(C + \overline{C})$$
$$F = AB + A\overline{B}$$
$$F = A(B + \overline{B})$$
$$F = A$$

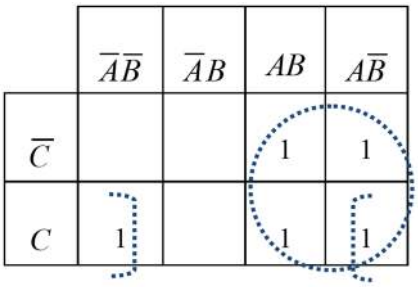
However, it is not the end of the simplification as there is one more 1 that is not paired (for $F = \overline{A}\,\overline{B}C$). Loops in K-maps can wrap around, so create a pair loop as shown in Figure 4.6(b). Variable A is in complemented and uncomplemented forms in the pair loop, so it will disappear leaving only $\overline{B}C$. So the resulting simplified expression will be $F = A + \overline{B}C$.

		00	01	11	10
		$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
0	$\overline{C}$			1	1
1	C	1		1	1

Figure 4.5: Three variable K-map for . $F = ABC + A\overline{B}C + \overline{A}\overline{B}C + AB\overline{C} + A\overline{B}\overline{C}$.

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$			1	1
C	1		1	1

(a)



(b)

Figure 4.6: Three variable K-map shown in Figure 4.5: (a) quad loop (b) quad with pair loop.

As another example, consider $F = \overline{A}\overline{B} + A\overline{B}\overline{C} + A\overline{B}C$. Since one of the terms, $\overline{A}\overline{B}$ has only two variables, it should be expanded to give $\overline{A}\overline{B} = \overline{A}\overline{B}(C + \overline{C}) = \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C}$. So $F = \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$. Now the K-map can be constructed as shown in Figure 4.7 and quad loop applied to give $F = \overline{B}$.

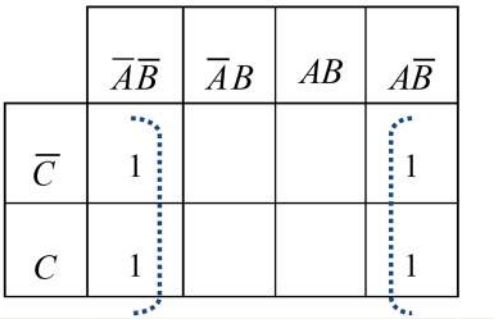


Figure 4.7: Three variable K-map for $F = \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$.

It can be verified that algebraic simplification also gives the same result:

$$F = \overline{A}\overline{B} + A\overline{B}\overline{C} + A\overline{B}C$$

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

$$F = \overline{A}\overline{B} + A\overline{B}$$

$$F = \overline{B}(A + A)$$

$$F = \overline{B}$$

4.3.3 Four variable K-map

Consider a logic expression with four variables:

$$F = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D$$
$$+ ABCD + A\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}BC\overline{D}$$

Figure 4.8 shows the K-map for this expression. With four variables, octet looping (with eight 1s) is possible. Note that loops should be as big as possible, so if there is a choice of two quad loops and one octet loop, then the octet loop should be created.

Only variable $\overline{C}$ remains from the octet loop as the other variables are in both complement and uncomplemented forms and hence disappear. There are two quad loops that give AD and $\overline{A}\overline{D}$ (wrapped around loop). The final expression is $F = AD + \overline{A}\overline{D} + \overline{C}$ ..

It should be obvious now that a pair loop removes one variable, a quad loop removes two variables while an octet loop removes three variables. In the example above, octet loop removed variables A, B and D.

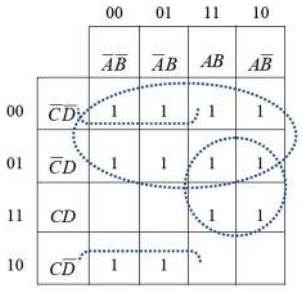


Figure 4.8: Four variable K-map.

4.3.4 Additional examples

Consider the truth table as in Table 4.7. For this example, let us obtain the simplified logic circuit diagram.

Table 4.7: Truth table for additional example 1

A	B	C	D	F	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	$\rightarrow F = \overline{A}\overline{B}CD$
0	1	0	0	0	
0	1	0	1	1	$\rightarrow F = \overline{A}B\overline{C}D$
0	1	1	0	0	
0	1	1	1	1	$\rightarrow F = \overline{A}BCD$
1	0	0	0	1	$\rightarrow F = A\overline{B}\overline{C}\overline{D}$
1	0	0	1	1	$\rightarrow F = A\overline{B}\overline{C}D$
1	0	1	0	1	$\rightarrow F = A\overline{B}C\overline{D}$
1	0	1	1	1	$\rightarrow F = A\overline{B}CD$
1	1	0	0	1	$\rightarrow F = AB\overline{C}\overline{D}$
1	1	0	1	1	$\rightarrow F = AB\overline{C}D$
1	1	1	0	1	$\rightarrow F = ABC\overline{D}$
1	1	1	1	1	$\rightarrow F = ABCD$

First, the logic expression should be obtained from the truth table and using it, K-map drawn (as shown in Figure 4.9). Next, we can obtain the simplified expression and with it draw the simplified logic circuit diagram as shown in Figure 4.10.

Logic expression:

$$F = \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BCD + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + ABC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D}$$

K-map:

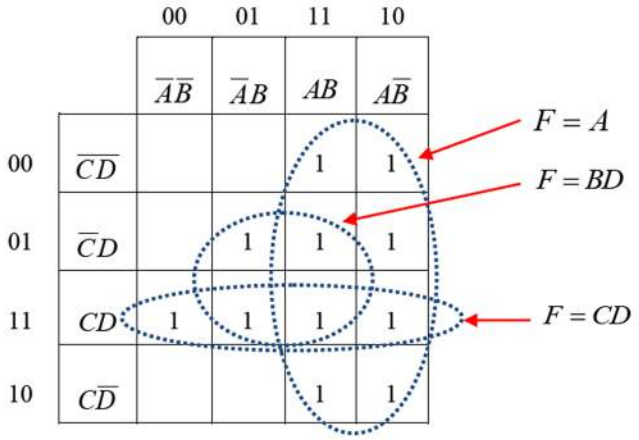


Figure 4.9: K-map for additional example 1.

Simplified expression: $F = A + BD + CD$.

Simplified logic circuit diagram:

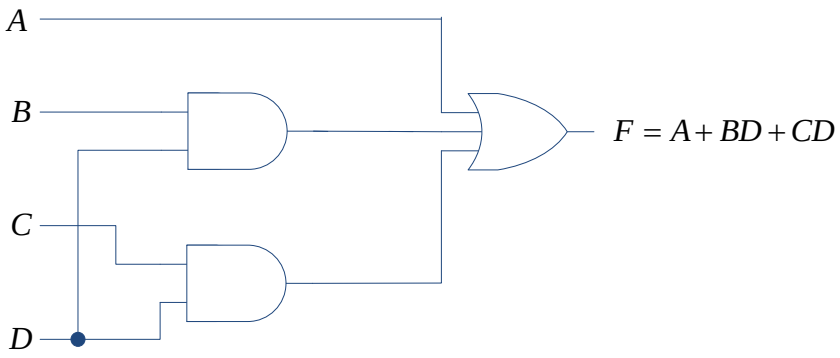


Figure 4.10: Simplified logic circuit diagram additional example 1.

As another example, consider a logic expression, $F = A\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}C\overline{D}$ and its corresponding K-map as shown Figure 4.11.

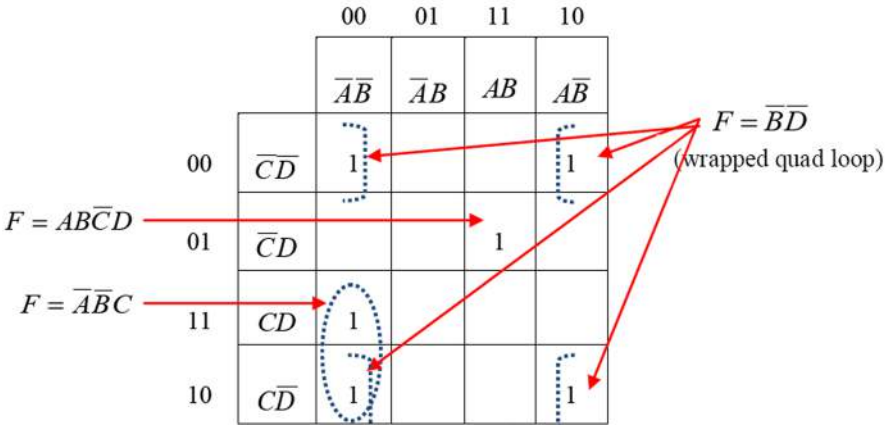


Figure 4.11: K-map for the additional example 2.

The wrapped around quad loop gives $\overline{B}\,\overline{D}$ while the pair loop gives $\overline{A}\,\overline{B}C$. There is a single 1 that can't be looped, so it remains as it is: $AB\overline{C}D$. So, the simplified expression is $F = AB\overline{C}D + \overline{A}\,\overline{B}C + \overline{B}\,\overline{D}$.

4.3.5 Don't care conditions

In digital logic design, we often encounter don't care conditions. These conditions are cases that won't occur in our design and hence the output can be set to any value (either 0 or 1). Don't care conditions are denoted using X in the truth tables and K-maps. For example, consider a seven segment display device as shown in Figure 4.12 that is commonly used to display hexadecimal characters.

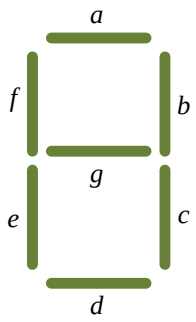


Figure 4.12: Seven segment display.

The device consists of light emitting diodes (LEDs)[7] that light up with different patterns to give the hexadecimal output as shown in Figure 4.13. Note that the hex characters A to F are normally displayed in a mixture of upper and lowercase to avoid ambiguity (for example differentiating D with 0, B with 8 etc).

7 Newer devices operate using liquid crystal technology.

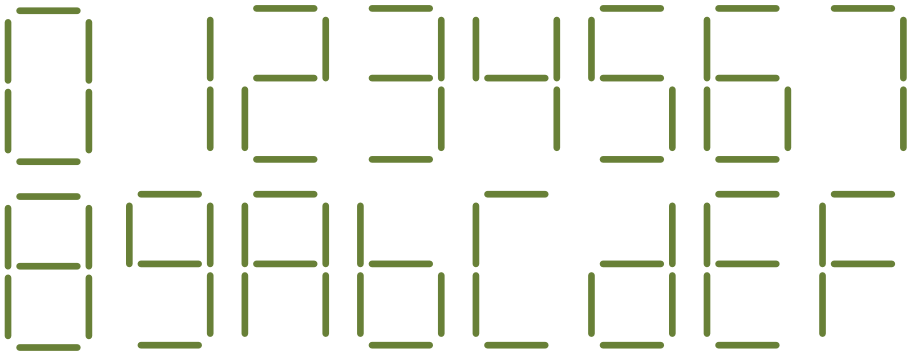


Figure 4.13: Hex characters displayed by the seven segment display.

Table 4.8 gives the character encodings for the seven LEDs (*a, b, ... ,g*), where a 1 denotes that the LED will be ON and a 0 denotes that the LED will be OFF. So to display numeral 0, LEDs *a, b, c, d, e,* and *f* will be turned on and LED *g* will be off. Similarly, to display character F, LEDs *a, e, f,* and *g* will be on while LEDs *b, c,* and *d* will be off.

Table 4.7: Character encodings for seven segment display LEDs

	LED						
Digit	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
A	1	1	1	0	1	1	1
b	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
d	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

Now, for the sake of discussing the don't care conditions, consider that we are going to use the seven segment display only to display the decimal numerals (i.e. 0 to 9). So, while designing the necessary wiring for the device, we can now ignore displays for the rest of the characters A to F. This situation will be denoted with X as in Table 4.8. Let us obtain the logic expression for LED *a*. To avoid confusion with the hex characters, we'll denote the variables as *P, Q, R,* and *S* instead of *A, B, C* and *D* as used earlier. Four variables (i.e. four inputs) are required since we have ten possible combinations.

Table 4.8: Seven segment display LED encoding for decimals (showing don't care conditions)

	LED						
Digit	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
A	X	X	X	X	X	X	X
b	X	X	X	X	X	X	X
C	X	X	X	X	X	X	X
d	X	X	X	X	X	X	X
E	X	X	X	X	X	X	X
F	X	X	X	X	X	X	X

Using the truth table, we can now construct the K-map as shown in Figure 4.14 (without considering don't care conditions) and Figure 4.15 (with don't care conditions).

Table 4.9: Truth table for LED *a*

Digit	*P*	*Q*	*R*	*S*	LED *a*
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	X
b	1	0	1	1	X
C	1	1	0	0	X
d	1	1	0	1	X
E	1	1	1	0	X
F	1	1	1	1	X

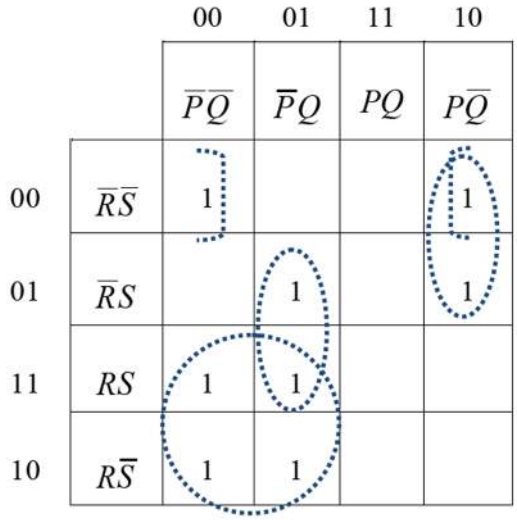


Figure 4.14: K-map for LED *a* without considering don't care conditions.

The simplified expression without considering don't care conditions is $LED_a = \overline{P}R + \overline{P}QS + P\overline{Q}\,\overline{R} + \overline{Q}\,\overline{R}\,\overline{S}$. Note that the solution is not unique as the wrapped around pair loop could also be formed for $\overline{P}\,\overline{Q}\,\overline{R}S$ and $\overline{P}Q\overline{R}S$ giving $\overline{P}\,\overline{Q}S$ instead of $\overline{Q}\,\overline{R}\,\overline{S}$ as shown for $P\overline{Q}\,\overline{R}S$ and $\overline{P}Q\overline{R}S$. With this, the simplified expression will be $LED_a = \overline{P}R + \overline{P}QS + P\overline{Q}\,\overline{R} + \overline{P}\,\overline{Q}\,\overline{S}$.

Now consider Figure 4.15 where the don't care conditions are accounted. Since X is either 0 or 1, we can assume it to be 1 and use in the looping procedures.

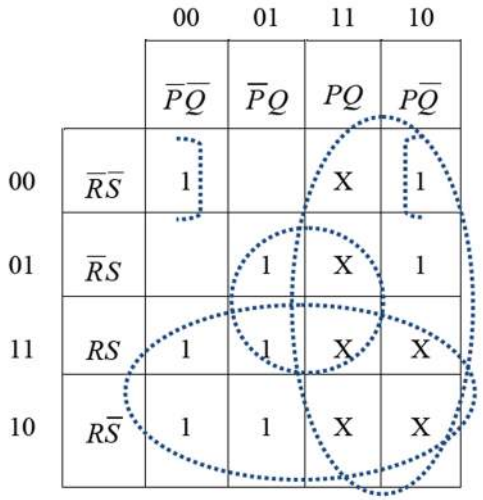


Figure 4.15: K-map for LED *a* (considering don't care conditions).

The simplified expression is now $LED_a = P + R + QS + \overline{Q}\overline{R}\overline{S}$ and it can be seen that the expression is made simpler by considering the don't care conditions.

As a final example for the chapter, let us obtain the logic expression for LED *b*. Table 4.10 gives the truth table and Figure 4.16 shows the K-map with don't care conditions. The simplified logic expression is $LED_b = RS + \overline{R}\overline{S} + \overline{Q} = \overline{R \oplus S} + \overline{Q}$. It should not be forgotten that the loops should be as big as possible.

Table 4.10: Truth table for LED *b*

Digit	*P*	*Q*	*R*	*S*	LED *b*
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
A	1	0	1	0	X
b	1	0	1	1	X
C	1	1	0	0	X
d	1	1	0	1	X
E	1	1	1	0	X
F	1	1	1	1	X

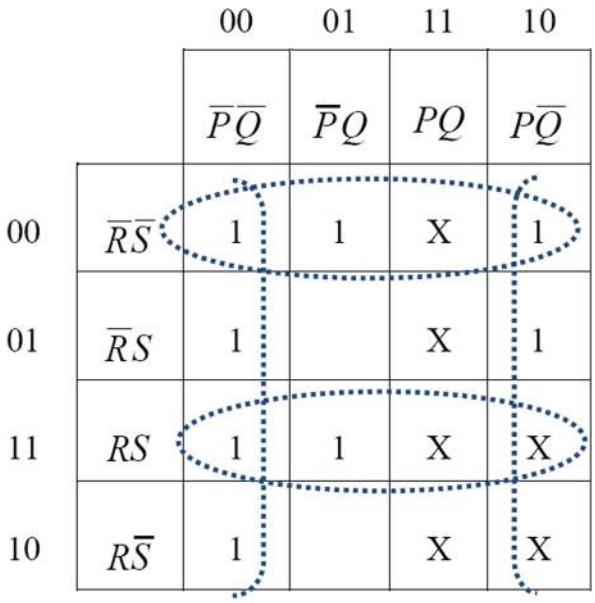


Figure 4.16: K-map for LED *b* with don't care conditions.