C H A P T E R I V

OTHER COMBINATIONAL LOGIC DESIGN TECHNIQUES

COMPOSITE MAPPING

When one explains the use of Karnaugh maps or the Quine-McCluskey technique for the reduction of Boolean functions there is the tacit assumption that the minterms of the function to be reduced are identified and directly available. When this is not true, one must first decompose* the function before beginning the reduction process. This decomposition may be avoided, in the case of Karnaugh map reduction, by using a composite mapping technique suggested by Smothers.[1]

Suppose we were given the Boolean function

$$f(a,b,c,d) = [(M_{15}M_5)a(b+c)] + m_{15} + m_8 + m_7 + m_3 \qquad (4\text{-}1)$$

---

*The term "decomposition" is used here as meaning the process of expanding reduced minterms or Maxterms into their constituent unreduced minterms or Maxterms. For example $f(a,b,c) = a + b$ decomposes (expands) to $f(a,b,c) = \bar{a}b\bar{c} + \bar{a}bc + ab\bar{c} + abc + a\bar{b}\bar{c} + a\bar{b}c$. The reader is also referenced to the expansion theorem T-10.

The usual procedure to reduce this function via a Karnaugh map would be to expand the term $T = [(M_1 \cdot M_5)a(b+c)]$ into a disjunctive normal form as follows:

$$T = (a+b+c+d)(\bar{a}+b+\bar{c}+d)a(b+c) \tag{4-2}$$

$$= (\cancel{ab}+a\bar{c}+\cancel{ad}+\cancel{\bar{a}b}+b+\cancel{b\bar{c}}+\cancel{bd}+\bar{a}c+\cancel{bc}+\cancel{cd}+\cancel{\bar{a}d}+\cancel{bd}+\cancel{\bar{c}d}+d)a(b+c)$$

$$= (a\bar{c}+\bar{a}c+b+d)(ab+ac)$$

$$= \cancel{abc}+ab+\cancel{abd}+\cancel{abc}+acd$$

$$T = ab + acd \tag{4-3}$$

Equation (4-1) may now be written in disjunctive normal form as

$$\therefore f(a,b,c,d) = ab+acd+abcd+a\bar{b}\bar{c}\bar{d}+\bar{a}bcd+\bar{a}\bar{b}cd \tag{4-4}$$

This equation is now ready for plotting on a four-variable Karnaugh map as shown in Fig. 4-1. The reduced form may be readily determined to be

$$f(a,b,c,d) = ab + cd + a\bar{c}\bar{d} \tag{4-5}$$

a b	0 0	1 0	1 1	0 1	d c
0 0			1		
0 1			1		
1 1	1	1	1	1	
1 0	1		1		

Figure 4-1 $f(a,b,c,d) = ab+cd+a\bar{c}\bar{d}$

The composite mapping technique permits one to avoid the algebra of decomposition by breaking the function to be reduced into terms which are Boolean products and terms which are Boolean sums and plotting these directly on their respective Karnaugh maps.

For our example we will let

$$f_1 = (a + b + c + d)(\bar{a} + b + \bar{c} + d) \qquad (4\text{-}6)$$

$$f_2 = ab + ac \qquad (4\text{-}7)$$

$$f_3 = abcd + a\bar{b}\bar{c}\bar{d} + \bar{a}bcd + \bar{a}\bar{b}cd \qquad (4\text{-}8)$$

$$f(a,b,c,d) = (f_1 \cdot f_2) + f_3 \qquad (4\text{-}9)$$

Since $f_1 = M_{15} \cdot M_5$ (4-10)

$$= \bar{m}_0 \cdot \bar{m}_{10} \qquad \text{since } M_\beta = \bar{m}_{2^n-1-\beta} \qquad (4\text{-}11)$$

The Karnaugh map for the term f_1 is given in Fig. 4-2 as

a b \ d c	0 0	1 0	1 1	0 1
0 0	0	1	1	1
0 1	1	1	1	1
1 1	1	1	1	1
1 0	1	1	1	0

Figure 4-2 Karnaugh map for the term f_1

since $\bar{m}_0 = \overline{\bar{a}\bar{b}\bar{c}\bar{d}}$ and $\bar{m}_{10} = \overline{a\bar{b}cd}$ (4-12)

The Karnaugh maps for the terms f_2 and f_3 are given in Figs. 4-3(a),(b).

a b \ d c	0 0	1 0	1 1	0 1
0 0	0	0	0	0
0 1	0	0	0	0
1 1	1	1	1	1
1 0	0	0	1	1

(a)

a b \ d c	0 0	1 0	1 1	0 1
0 0			1	
0 1			1	
1 1			1	
1 0	1			

(b)

Figure 4-3 Karnaugh maps for f_2(a) and f_3(b)

The <u>composite</u> map for $(f_1 \cdot f_2)$ is derived in Fig. 4-4.

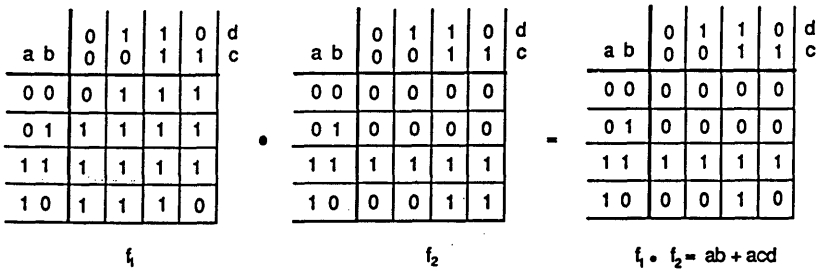


Figure 4-4 Composite map derivation for $f_1 \bullet f_2$

The composite map for $f(a,b,c,d) = f_1 \cdot f_2 + f_3$ is derived in Fig. 4-5.

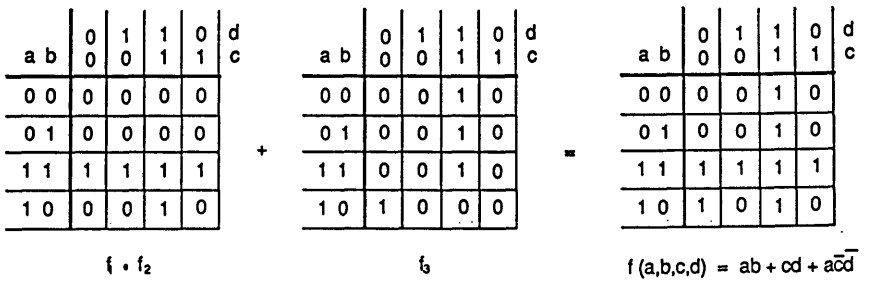


Figure 4-5 Composite map derivation for $f(a,b,c,d) = ab + cd + a\bar{c}\bar{d}$

This method would appear to require a great deal of map construction, however in practice a single map will suffice. Figure 4-6 shows this single map implementation. Note that the shaded cells of this Fig. 4-6 are identical to the map of Fig. 4-5.

The only subtilty in constructing the composite map is the treatment of Maxterms. Algorithmically one simply complements all literals and plots the result as a minterm and as zero.

a	b	0 0	1 0	1 1	0 1	d c
0	0	(0.0)+0	(1.0)+0	(1.0)+1	(1.0)+0	
0	1	(1.0)+0	(1.0)+0	(1.0)+1	(1.0)+0	
1	1	(1.1)+0	(1.1)+0	(1.1)+1	(1.1)+0	
1	0	(1.0)+1	(1.0)+0	(1.1)+0	(0.1)+0	

Figure 4-6 A single composite map construction for equation (4-9)
$f(a,b,c,d) = (f_1 \cdot f_2) + f_3$

For example, the Maxterm $M_5 = \bar{a} + b + \bar{c} + d$ is plotted as $a\bar{b}c\bar{d}$ as a ZERO. Equations (10, 11, and 12) legitimize this procedure.

This composite mapping technique, as with Karnaugh maps in general, has a practical limit of six variables.

Input Minimization Using Multi-Level Factoring[2]

This technique is based on the concept of minterm isolation (Theorem T-8), equations (1-10a) and (b). Here one artificially "adds" minterms to a Karnaugh map to achieve greater reduction and concurrently cancels these minterms with equation (1-10b). Where the designer confronts the problem of "adding" minterms he usually has several choices, some of which produce superior reductions. He must therefore explore all possibilities and select the one which

results in the fewest number of gated inputs. This method results in one or more additional levels of gating but this added complexity is hopefully balanced by a decrease in gated inputs. This method was first proposed by Levine.[2] Two examples will be worked to illustrate the method.

First consider the implementation of the function

$$f(a,b,c,d) = \sum 0, 2, 3, 15 \qquad (4\text{-}12)$$

The Karnaugh map for this function is shown in Fig. 4-7(a).

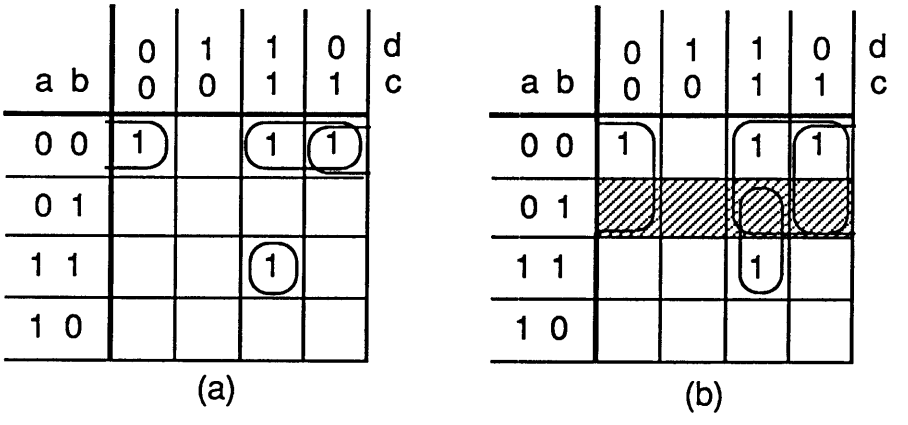


Figure 4-7 (a) Karnaugh map for f(a,b,c,d) = 0,2,3,15
(b) Modified Karnaugh map

The map (4-7a) reduction yields

$$f(a,b,c,d) = \bar{a}\bar{b}\bar{d} + \bar{a}\bar{b}c + abcd \qquad (4\text{-}13)$$

This function requires four gates and thirteen gated inputs as shown in Fig. 4-8.

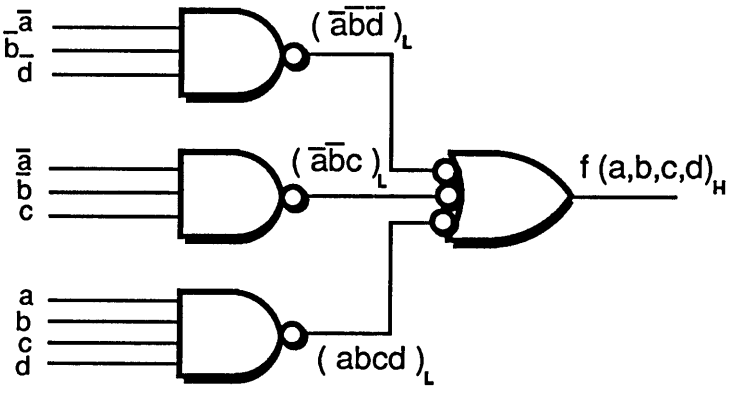


Figure 4-8 NAND implementation of equation (4-13)

If we now assume 1's in minterms m_4, m_5, m_6, and m_7 the Karnaugh map is altered as shown in Fig. 4-7(b). The reduction of this map yields

$$f(a,b,c,d) = (\bar{a}\bar{d} + \bar{a}c + bcd)(\overline{\bar{a}b}) \qquad (4\text{-}14)$$

The parenthetical term $(\bar{a}\bar{d} + \bar{a}c + bcd)$ is derived as indicated by the map groupings and the term $(\overline{\bar{a}b})$ is the term which cancels minterms m_4, m_5, m_6, m_7. We have invoked T-8 that says $(\overline{\bar{a}b})$ ANDED with minterms m_0, m_2, m_3 and m_{15} leaves them unchanged and $(\overline{\bar{a}b})$ ANDED with minterms m_4, m_5, m_6, and m_7 reduces each to zero. A little algebra will show this to be true.

$$(\sum 4,5,6,7)\overline{\bar{a}b} = (\bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd)(a+\bar{b}) \equiv 0$$

since $m_i \bar{m}_j = 0$ if $i = j$ (T-8a)

and $(\sum 0,2,3,15)\overline{\bar{a}\bar{b}} = (\bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}d + abcd)(a+\bar{b})$

$= abcd + \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}d$

since $m_i \bar{m}_j = m_i$ if $i \neq j$ (T-8a)

Figure 4-9 is the logic diagram for equation (4-14).

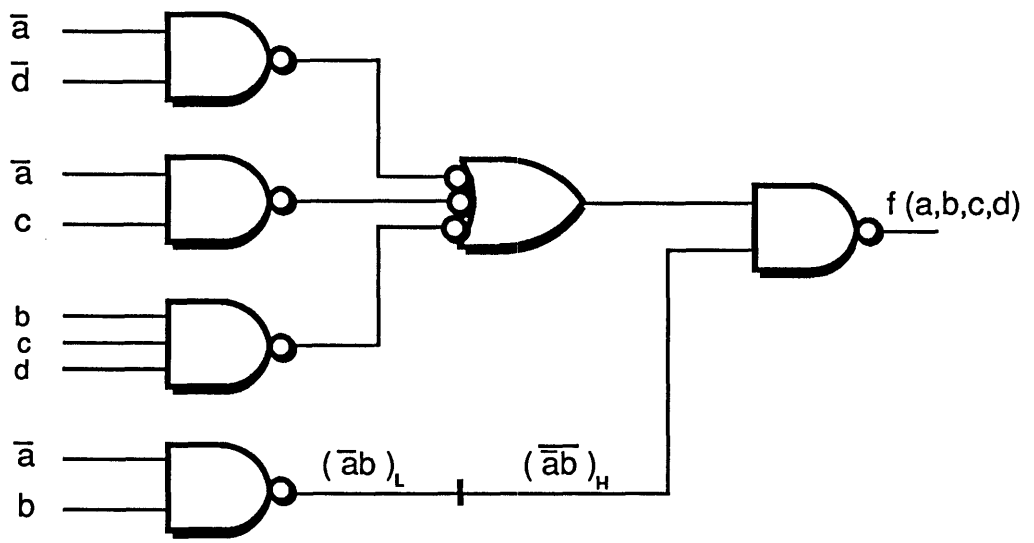
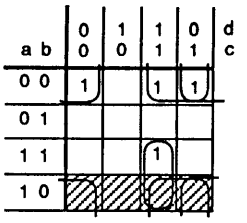


Figure 4-9 Logic diagram for $f(a,b,c,d) = (\bar{a}\bar{d}+\bar{a}c+bcd)(\overline{\bar{a}\bar{b}})$

This solution has six gates and fourteen gated inputs.

Figure 4-10 illustrates four other possible solutions using this same approach. The last solution is obviously the best because it requires only eleven gated inputs. This solution's logic diagram is shown in Fig. 4-10(d).

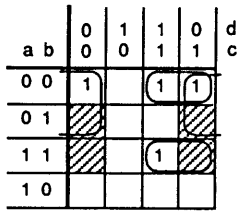


$= a\bar{b}$

$f(a,b,c,d) = (\bar{b}c + \bar{b}\bar{d} + acd)(\overline{a\bar{b}})$ (4-15)

Six gates and fourteen gated input

(a)

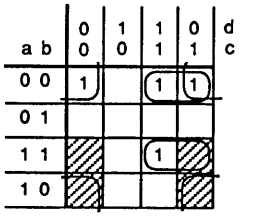


$= b\bar{d}$

$f(a,b,c,d) = (\bar{a}\bar{d} + \bar{a}\bar{b}c + abc)(\overline{b\bar{d}})$ (4-16)

Six gates and fifteen gated inputs

(b)

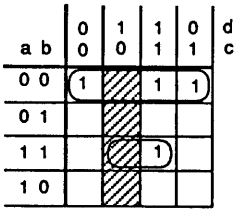


$= a\bar{d}$

$f(a,b,c,d) = (\bar{b}\bar{d} + abc + \bar{a}\bar{b}c)(\overline{a\bar{d}})$ (4-17)

Six gates and fifteen gated inputs

(c)

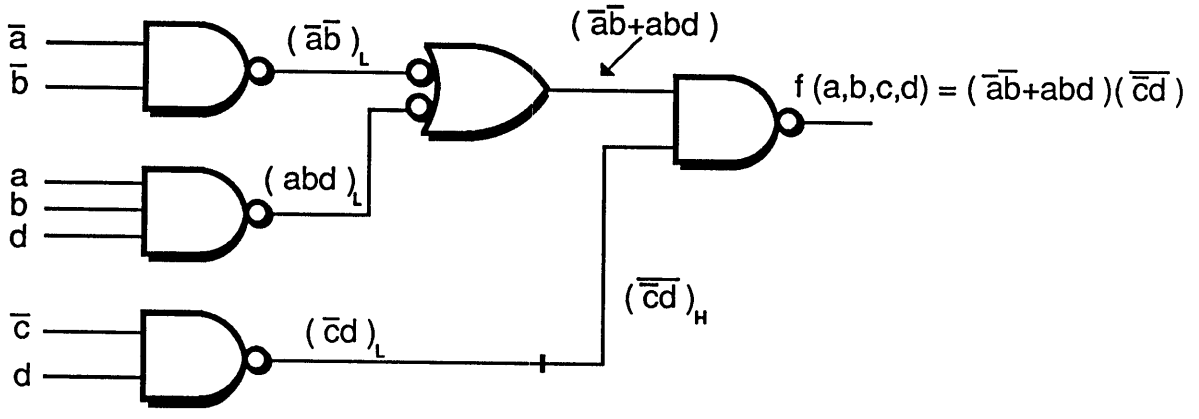


$= \bar{c}d$

$f(a,b,c,d) = (\bar{a}\bar{b} + abd)(\overline{\bar{c}d})$ (4-18)

five gates and eleven gated inputs

(d)



(e)

Figure 4-10 Input minimization of equation (4-14) as given by equation (4-18)

Before concluding this example it should be noted that equation (4-13) could be factored to yield

$$f(a,b,c,d) = \bar{a}\bar{b}(\bar{d} + c) + abcd$$

This solution has five gates and twelve gated inputs. On the basis of the complexity function (equation 2-5), equation (4-18) of Fig. 4-10(d) is the preferred form.

A second example, of slightly greater complexity, will be solved with minimal comment. Here, we will find the form of the function

$$f(a,b,c,d) = \sum 2,3,7,8,10,11,12,13,14,15 \qquad (4\text{-}19)$$

which will yield the simplest logical structure. The Karnaugh map of equation (4-19) is given in Fig. 4-11.

a b \ d c	0 0	1 0	1 1	0 1
0 0			1	1
0 1			1	
1 1	1	1	1	1
1 0	1		1	1

Figure 4-11 Karnaugh map of equation (4-19)

The direct reduction of this unmodified map yields

$$f(a,b,c,d) = ab + a\bar{d} + cd + \bar{b}c \qquad (4\text{-}20)$$

which implies five gates and twelve gated inputs. Equation (4-20) may be factored to yield

$$f(a,b,c,d) = a(b + \bar{d}) + c(\bar{b} + d) \qquad (4\text{-}21)$$

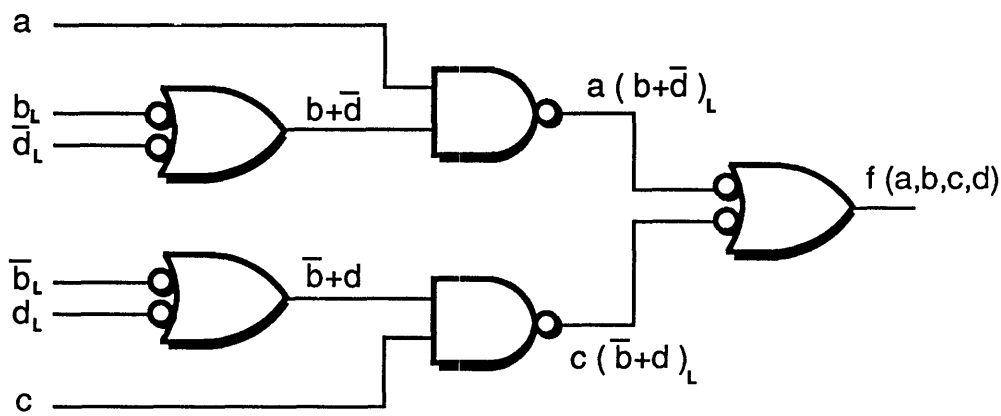


Figure 4-12 NAND implementation of equation (4-21)

We will now investigate map modifications and the resulting implied logical structures. Figures 4-13 thru 4-18 illustrate various modifications of the basic map for equation (4-19) given in Fig. 4-11. These modifications and their resulting logic diagrams are self-evident and shown without further comment.

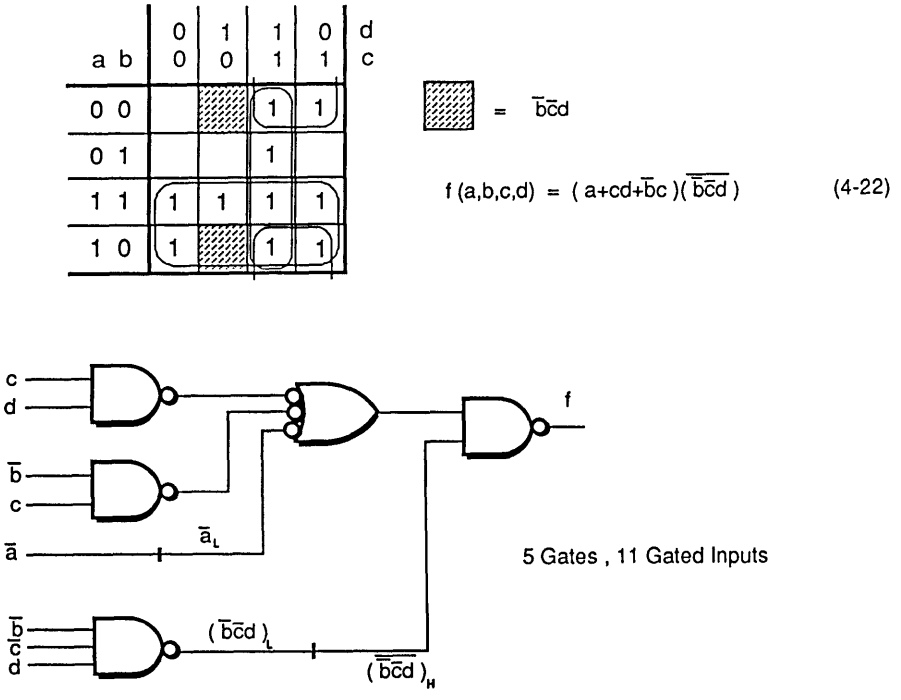


Figure 4-13 NAND implementation of equation (4-22)

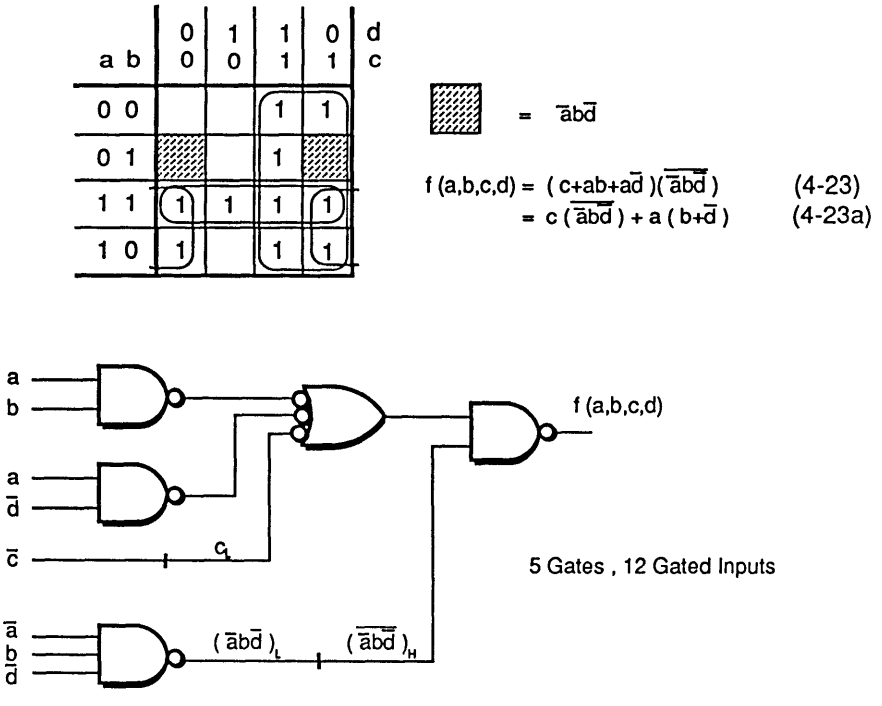


Figure 4-14 NAND implementation of equation (4-23)

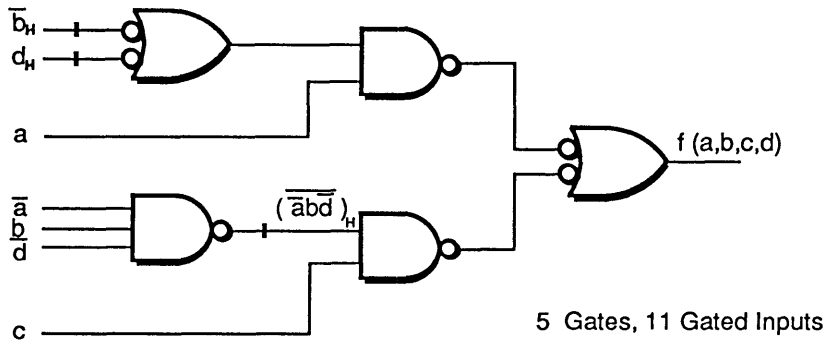
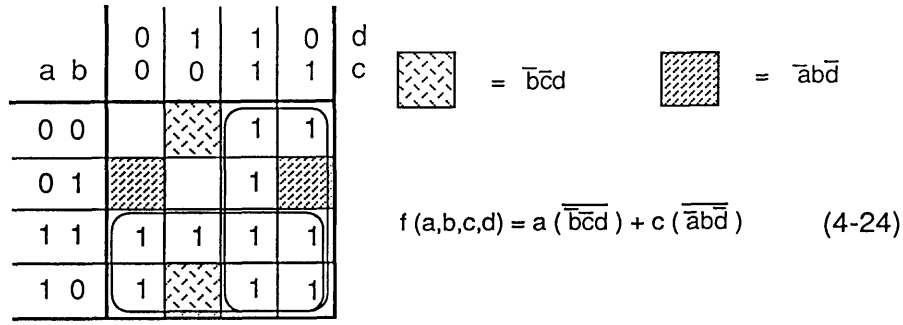


Figure 4-15 NAND implementation of equation (4-23a)



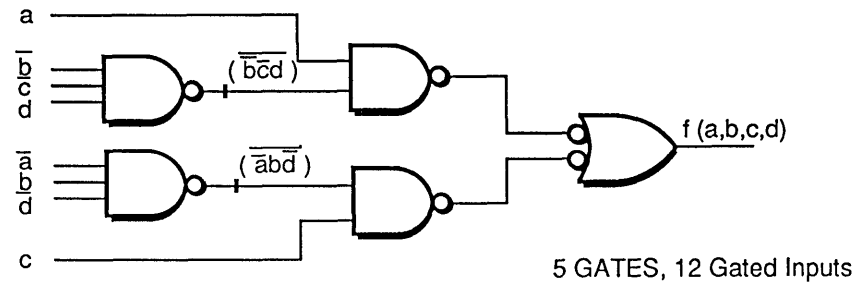


Figure 4-16 NAND implementation of equation (4-24)

a b \ d c	0 0	0 1	1 1	1 0
0 0			1	1
0 1			1	
1 1	1	1	1	1
1 0	1		1	1

$= \bar{a}\,\bar{c}$

$$f(a,b,c,d) = ab + cd + \bar{b}\bar{d}\,(\overline{\bar{a}\bar{c}}) \qquad (4\text{-}25)$$

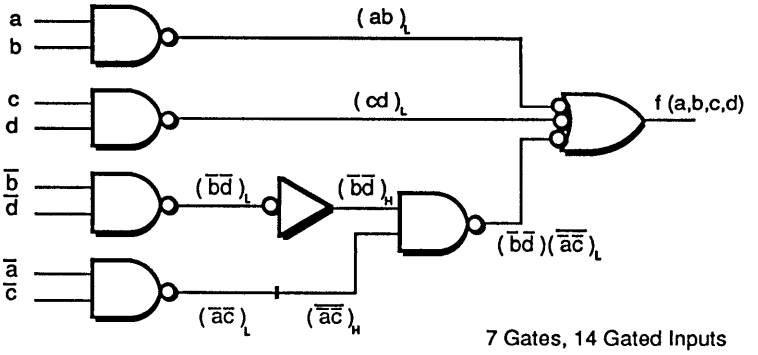


Figure 4-17 NAND implementation of equation (4-25)

a b \ d c	0 0	0 1	1 1	1 0
0 0			1	1
0 1			1	
1 1	1	1	1	1
1 0	1		1	1

$= \bar{a}\bar{c}\bar{d}$ $= \bar{a}b\bar{d}$

$$f(a,b,c,d) = (ab+c+\bar{d})(\overline{\bar{a}\bar{c}\bar{d}})(\overline{\bar{a}b\bar{d}}) \qquad (4\text{-}26)$$

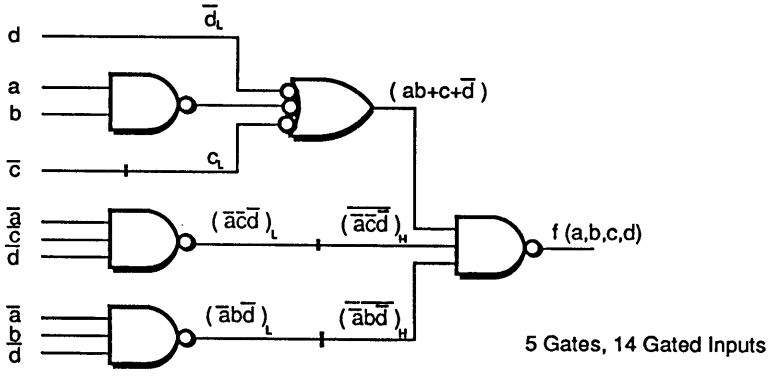


Figure 4-18 NAND implementation of equation (4-26)

One may conclude that implementing equation (4-21) minimizes the number of gated inputs. The reader is cautioned that this technique usually results in multi-level (beyond two) gating. The added gate delays might require running the system clock at a lower frequency. Note that in this second example, the "best" solution was three-level by virtue of factoring of the direct Karnaugh map solution.

DECOMPOSITION

Decomposition is a reduction technique wherein a Boolean function is broken down (decomposed) into smaller (simpler) functions and then finally combining these smaller functions to form the original logical equivalent. The idea of decomposition can be best presented with an example.

Consider the four-variable Boolean function

$$f(a,b,c,d) = \sum 1, 6, 7, 10, 11, 13 \qquad (4\text{-}27)$$

$$= \bar{a}\bar{b}\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}c\bar{d} + a\bar{b}cd + ab\bar{c}d$$

The Karnaugh map of this function is given in Fig. 4-19.

a b	0 0	1 0	1 1	0 1	d c
0 0		1			
0 1			1	1	
1 1		1			
1 0			1	1	

Figure 4-19 Karnaugh map for equation (4-27)

A two level reduction from this map yields

$$f(a,b,c,d) = \bar{a}bc + a\bar{b}c + \bar{a}\bar{b}\bar{c}d + ab\bar{c}d \qquad (4\text{-}28)$$

and its implementation is shown in Fig. 4-20.

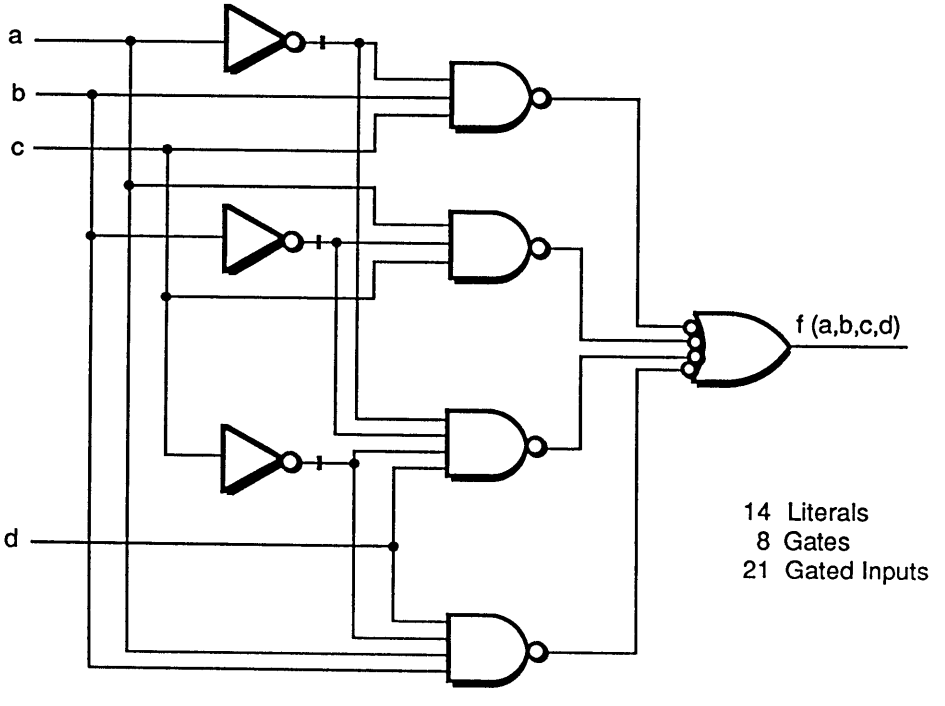
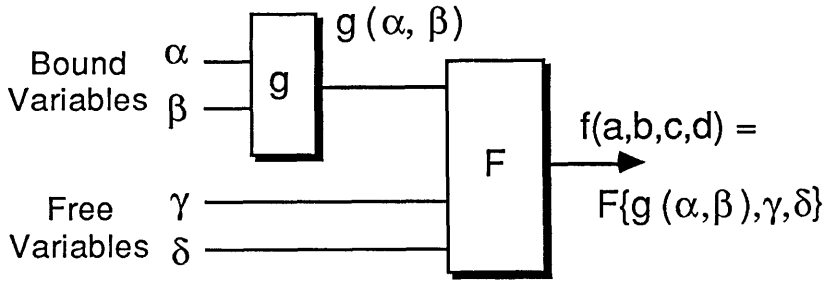
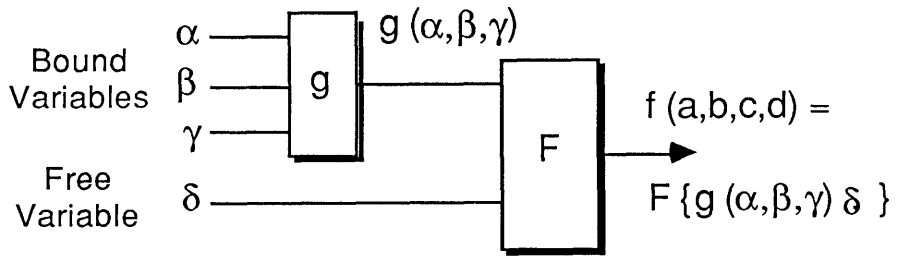


Figure 4-20 NAND implementation of equation (4-28)

Now let us look at the same problem with the idea of decomposing it to a form which looks like those shown in Fig. 4-21 (a) or (b).



(a)



(b)

Figure 4-21 Possible decomposition of f (a,b,c,d)

For the case of four-variables (a,b,c,d), α, β, γ and δ may each represent any one of these four-variables. Thus the two generalized decompositions of Fig. 4-21 actually represent ten possible logical structures of the same equation. The outputs of either form (a) or (b) must equal 0, 1, g or $\bar{g}$ for a valid partitioning or decomposition of the original function.

For four-variables, the ten specific possible logical structures are shown in Fig. 4-22.

We will presently show that all of the possible logic structures shown in Fig. 4-22 do not satisfy the requirement that the output f be equal to 0, 1, g or $\bar{g}$ for values of the free variables (γ and δ). Let us now test each of the ten logical structures of Fig. 4-22.

For (a) refer to Fig. 4-19 K-map

$$\left.\begin{aligned} f(a,b,0,0) &= 0 \\ f(a,b,0,1) &= \bar{a}\bar{b} + a\bar{b} \\ f(a,b,1,0) &= \bar{a}b + a\bar{b} \\ f(a,b,1,1) &= \bar{a}b + a\bar{b} \end{aligned}\right\} \quad (4\text{-}29a)$$

If we let $\bar{g}(a,b) = \bar{a}\bar{b} + ab$, then

$$\left.\begin{aligned} f(a,b,0,0) &= 0 \\ f(a,b,0,1) &= \bar{g}(a,b) \\ f(a,b,1,0) &= g(a,b) \\ f(a,b,1,1) &= g(a,b) \end{aligned}\right\} \quad (4\text{-}30)$$

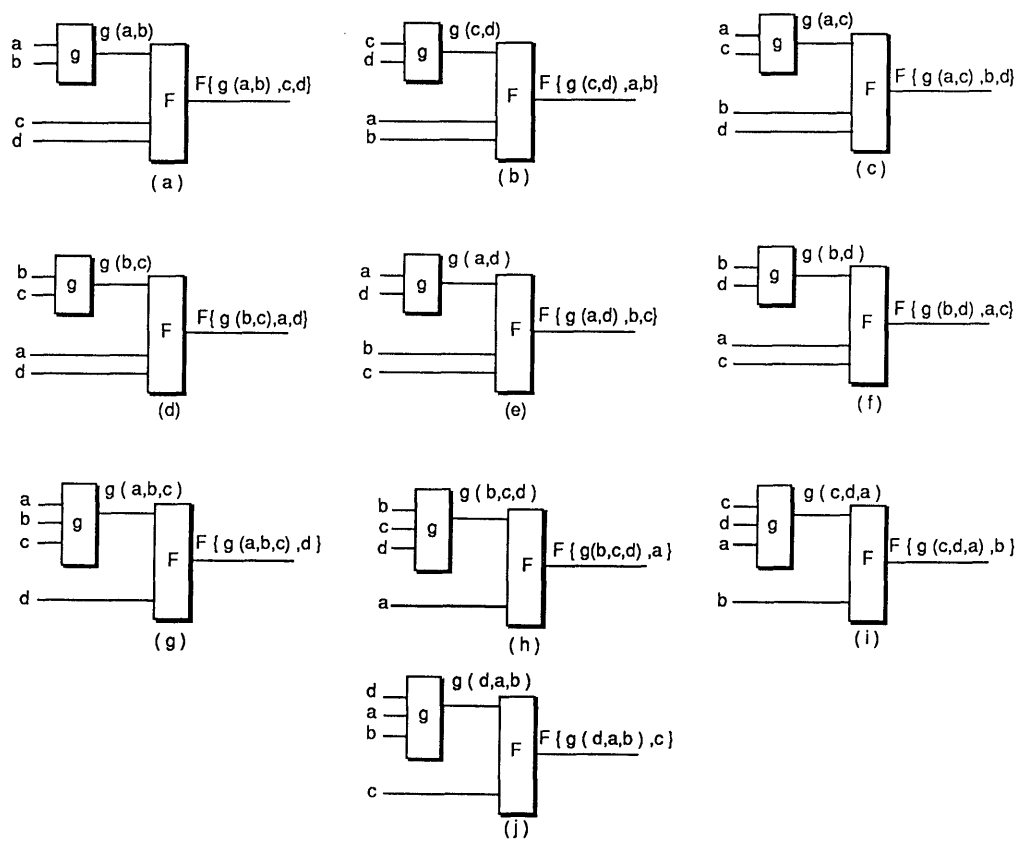


Figure 4-22 Possible decompositions of a four-variable function, f (a,b,c,d)

The output requirement, f = 0 or 1 or $\bar{g}$ or g is satisfied and the equation f(a,b,c,d) may be partitioned as indicated in Fig. 4-22(a).

$$f(a,b,c,d) = (0)\bar{c}\bar{d} + \bar{g}(a,b)\bar{c}d + g(a,b)c\bar{d} + g(a,b)cd \quad (4\text{-}31)$$
$$= \bar{g}(a,b)\bar{c}d + g(a,b)c\{\bar{d} + d\}$$
$$= \bar{g}(a,b)\bar{c}d + g(a,b)c$$

but $\bar{g}(a,b) = \bar{a}\bar{b} + ab$ and $g(a,b) = \bar{a}b + a\bar{b}$

$$\therefore f(a,b,c,d) = (\bar{a}\bar{b} + ab)\bar{c}d + (\bar{a}b + a\bar{b})c \quad (4\text{-}32)$$

Equation (4-32) may be implemented in NAND logic as shown in Fig. 4-23.

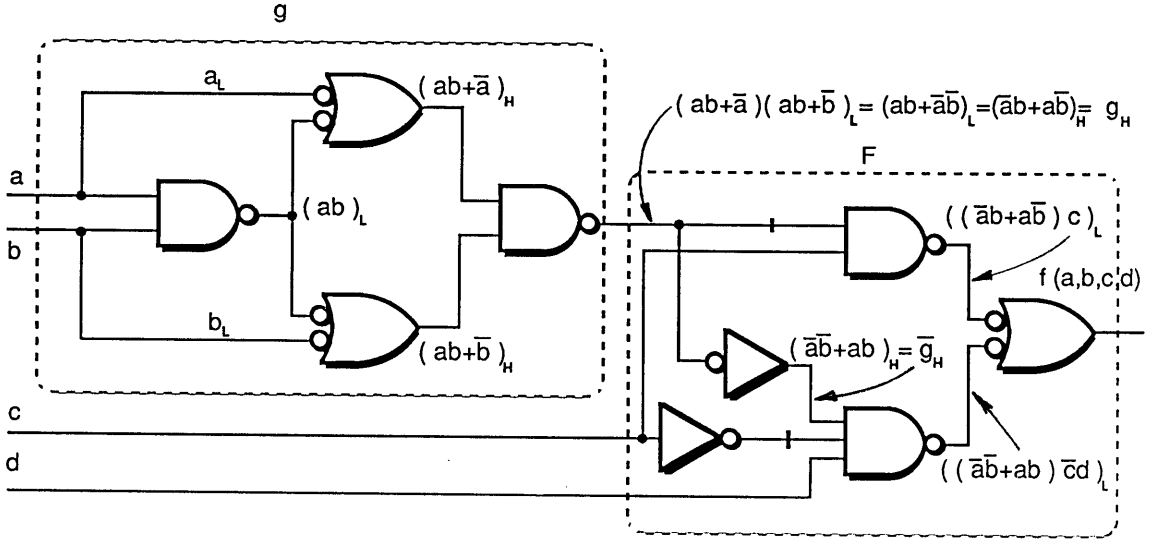


Figure 4-23 NAND implementation of
f (a,b,c,d) = F{g (a,b) , c,d} = $(\bar{a}b+a\bar{b})c+(\bar{a}\bar{b}+ab)\bar{c}d$

The complexity of this circuit should be compared with that of Fig. 4-20. Note that there are significantly fewer literals in the decomposed solution however because of the multilevel nature of the decomposed solution (Fig. 4-23) the circuit could be restrictive in operating speed due to the serial gate delays.

Continuing with the evaluation of structures (b) through (j) of Fig. 4-22 we find

For (b)

$$\left.\begin{aligned} f(0,0,c,d) &= \bar{c}d \\ f(0,1,c,d) &= c \\ f(1,0,c,d) &= c \\ f(1,1,c,d) &= \bar{c}d \end{aligned}\right\} \quad (4\text{-}29b)$$

Since $\bar{c}d$ and c are not complements the test fails and f(a,b,c,d) can not be realized with structure (b) of Fig. 4-22.

For (c)

$$\left.\begin{aligned} f(a,0,c,0) &= ac \\ f(a,0,c,1) &= \bar{a}\bar{c} + ac \\ f(a,1,c,0) &= \bar{a}c \\ f(a,1,c,1) &= \bar{a}c + a\bar{c} \end{aligned}\right\} \quad (4\text{-}29c)$$

For (d)

$$\left.\begin{aligned} f(0,b,c,0) &= bc \\ f(0,b,c,1) &= bc + \bar{b}\bar{c} \\ f(1,b,c,0) &= \bar{b}c \\ f(1,b,c,1) &= \bar{b}c + b\bar{c} \end{aligned}\right\} \quad (4\text{-}29d)$$

For (e)

$$\left.\begin{array}{l} f(a,0,0,d) = \bar{a}d \\ f(a,0,1,d) = a \\ f(a,1,0,d) = ad \\ f(a,1,1,d) = \bar{a} \end{array}\right\} \quad (4\text{-}29e)$$

For (f)

$$\left.\begin{array}{l} f(0,b,0,d) = \bar{b}d \\ f(0,b,1,d) = b \\ f(1,b,0,d) = b\bar{d} \\ f(1,b,1,d) = \bar{b} \end{array}\right\} \quad (4\text{-}29f)$$

Structures (c), (d), (e) and (f) of Fig. 4-22 can not realize $f(a,b,c,d) = \sum 1,6,7,10,11,13$ because their f outputs do not meet the test requiring them to be 0, 1, g or $\bar{g}$.

For the possible decompositions given by structures (g) of Fig. 4-22 we find

For (g)

$$\left.\begin{array}{l} f(a,b,c,0) = \bar{a}bc + a\bar{b}c \\ f(a,b,c,1) = \bar{a}bc + a\bar{b}c + \bar{a}\bar{b}\bar{c} + ab\bar{c} \end{array}\right\} \quad (4\text{-}29g)$$

For (h)

$$\left.\begin{array}{l} f(0,b,c,d) = bc + \bar{b}\bar{c}d \\ f(1,b,c,d) = \bar{b}c + b\bar{c}d \end{array}\right\} \quad (4\text{-}29h)$$

For (i)

$$\left.\begin{array}{l} f(a,0,c,d) = ac + \bar{a}\bar{c}d \\ f(a,1,c,d) = \bar{a}c + a\bar{c}d \end{array}\right\} \quad (4\text{-}29i)$$

For (j)

$$\left.\begin{aligned} f(a,b,0,d) &= \bar{a}\bar{b}d + abd \\ f(a,b,1,d) &= a \end{aligned}\right\} \quad (4\text{-}29j)$$

None of the configurations meets the test for f. Therefore the function can not be realized with structure (g), (h), (i) or (j) of Fig. 4-22 which are all of the form $F\{g(\alpha,\beta,\gamma),\delta\}$.

A much more convenient and rapid way of determining possible decompositions is to use decomposition maps. Here, Karnaugh maps are constructed for the function to be decomposed for each of the configurations as shown in Fig. 4-22. Two important points should be made here regarding the construction of these maps.

1. The designated free variables are assigned to the map rows and the designated bound variables are assigned to the map columns.
2. The minterm numerical entries are based on the bit weights a = 8, b = 4, c = 2 and d = 1 regardless of their position on the map structure. For example, map (b) of Fig. 4-24 is in conventional K-map structure where m_{13} is at the intersection of the free variables a = 1 and b = 1 with bound variables c = 0 and d = 1. In contrast, for map (a) where the free variables are c and d, m_{13} is at the interesection of free variables c = 0 and d = 1 with bound variables a = 1 and b = 1.

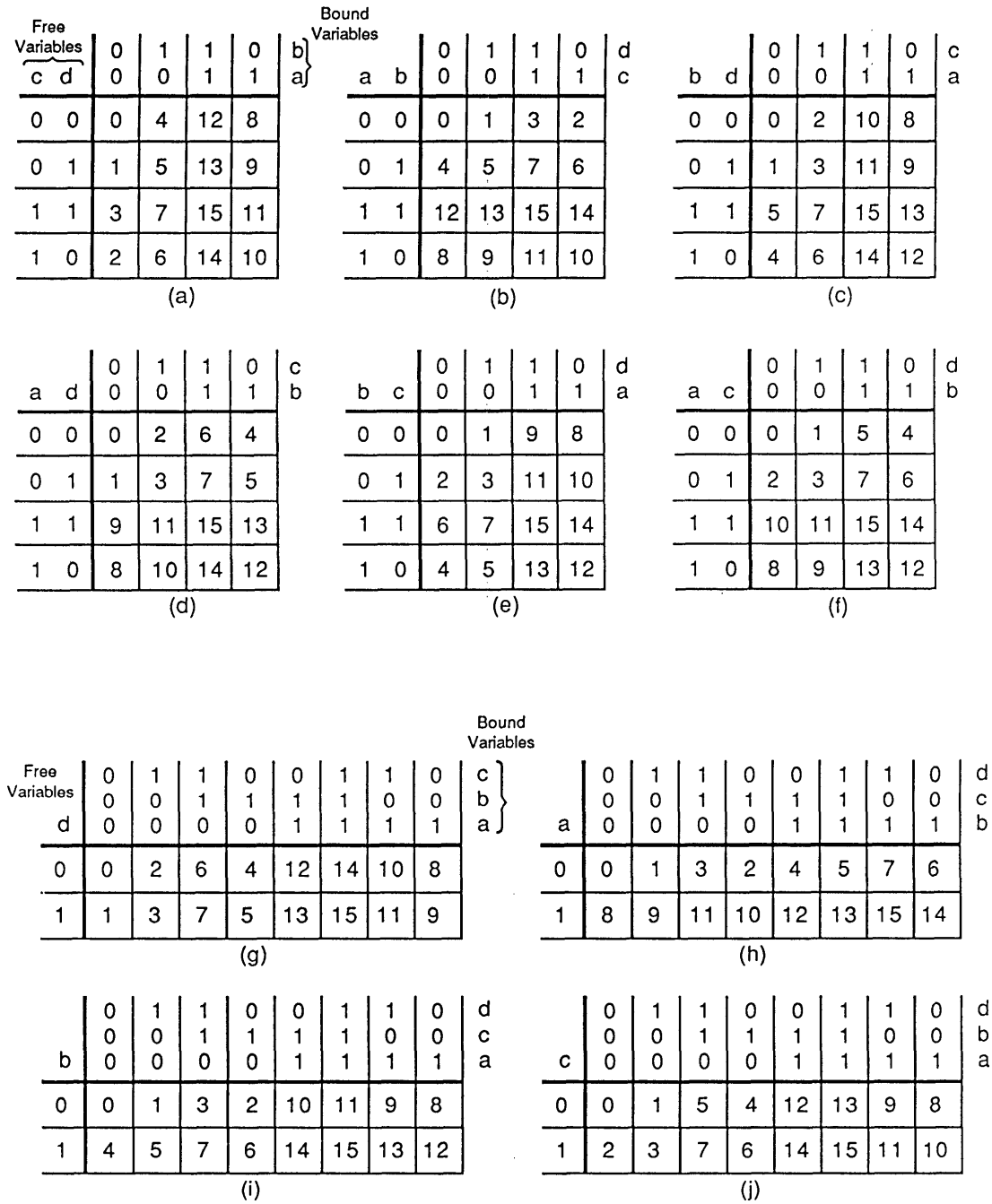


Figure 4-24 Decomposition maps for four-variable functions

These maps are used to readily determine which configurations (if any) of Fig. 4-22 yield decomposable forms of the subject function. To use the maps one merely circles the function's minterms on every map (10 maps for a four variable function) and then examines each map, column by column for "column multiplicity". Column multiplicity is a measure of the number of different column patterns on a given map. From a moment's contemplation it is obvious that if this multiplicity is greater than <u>two</u>, then the value of f does not satisfy the requirement that it equal 0, 1, g (bound variables) or $\bar{g}$ (bound variables).

Returning to the example function f(a,b,c,d) = $\sum$ 1, 6, 7, 10, 11, 13 of equation (4-27) the maps are as shown in Fig. 4-25.

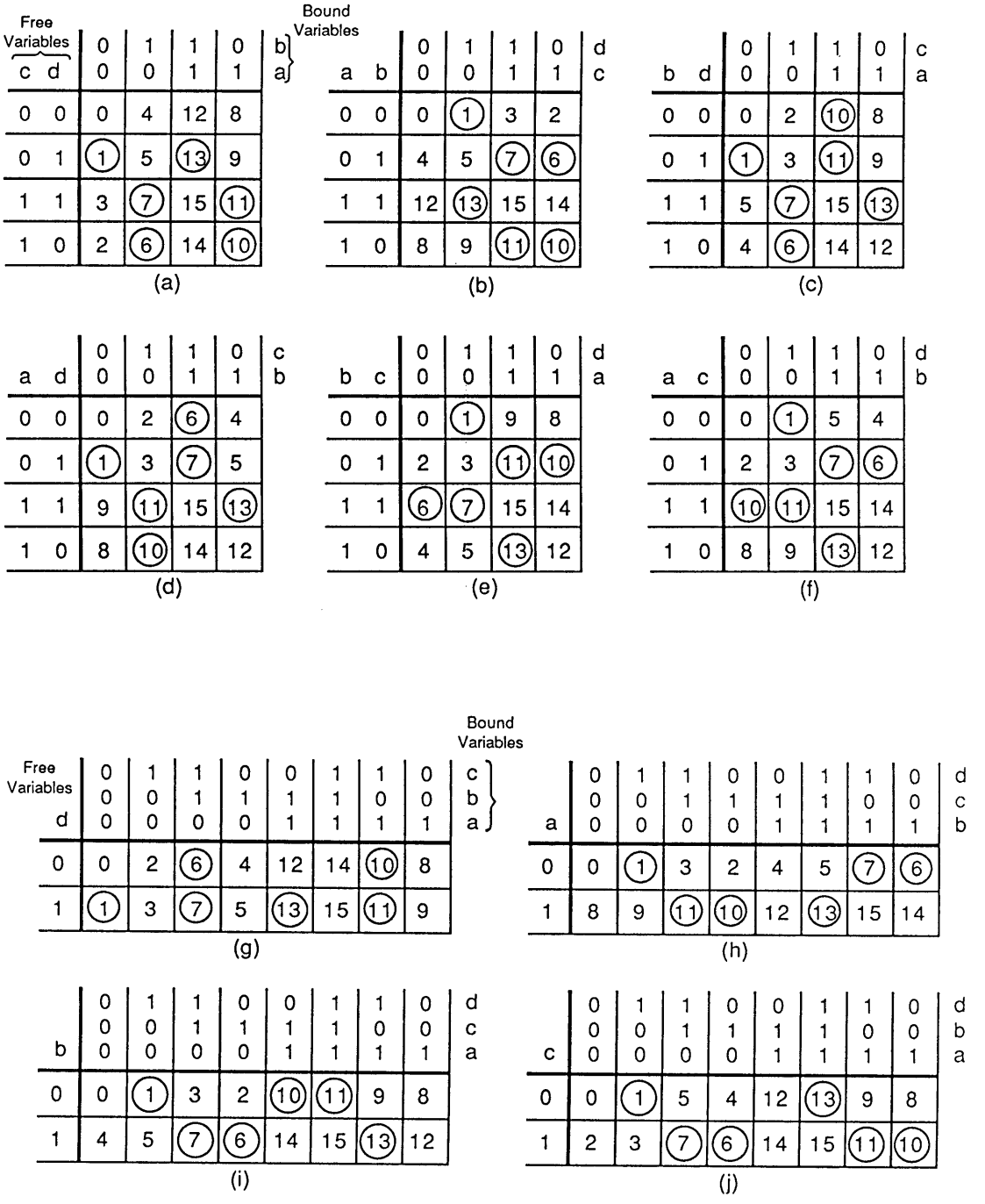


Figure 4-25 Decomposition maps for four-variable functions

It is immediately obvious that only map (a) of Fig. 4-25 has a column multiplicity of two or less. This map is redrawn in Fig. 4-26 and examined row by row for the 0, 1, g or $\bar{g}$ test.

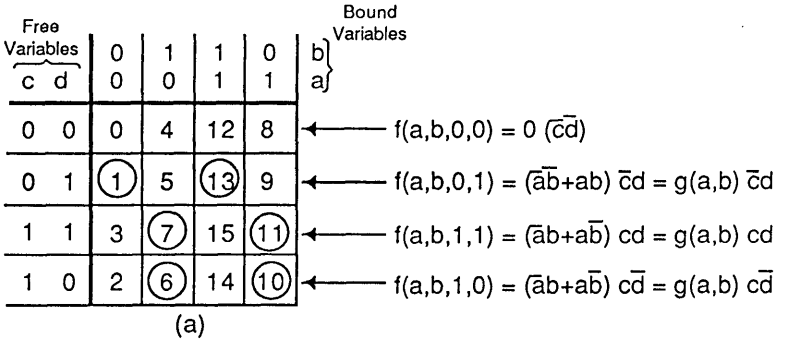


Figure 4-26 Examination of map (a) of Fig 4-25 to show that F{g(a,b),c,d} meets the 0,1,g or $\bar{g}$ requirement for partitioning

This procedure of plotting the function on preconstructed K-maps for all possible partitions and searching for maps with column multiplicities of two or less is the most preferred and rapid procedure for searching for possible functional partitions - once the basic decomposition process is understood. Remember, all Boolean functions cannot be decomposed while some may be decomposed into several alternative logical structures.

As a final example on this subject of decomposition, consider the function

$$f(a,b,c,d) = \sum 2, 3, 6, 7, 9, 10, 14, 15 \qquad (4\text{-}33)$$

The K-map set (10 maps) of possible decompositions is shown in Fig. 4-27. Here we immediately observe that maps (a), (e), (f), and (j) have column multiplicities of two. Possible partitions are therefore:

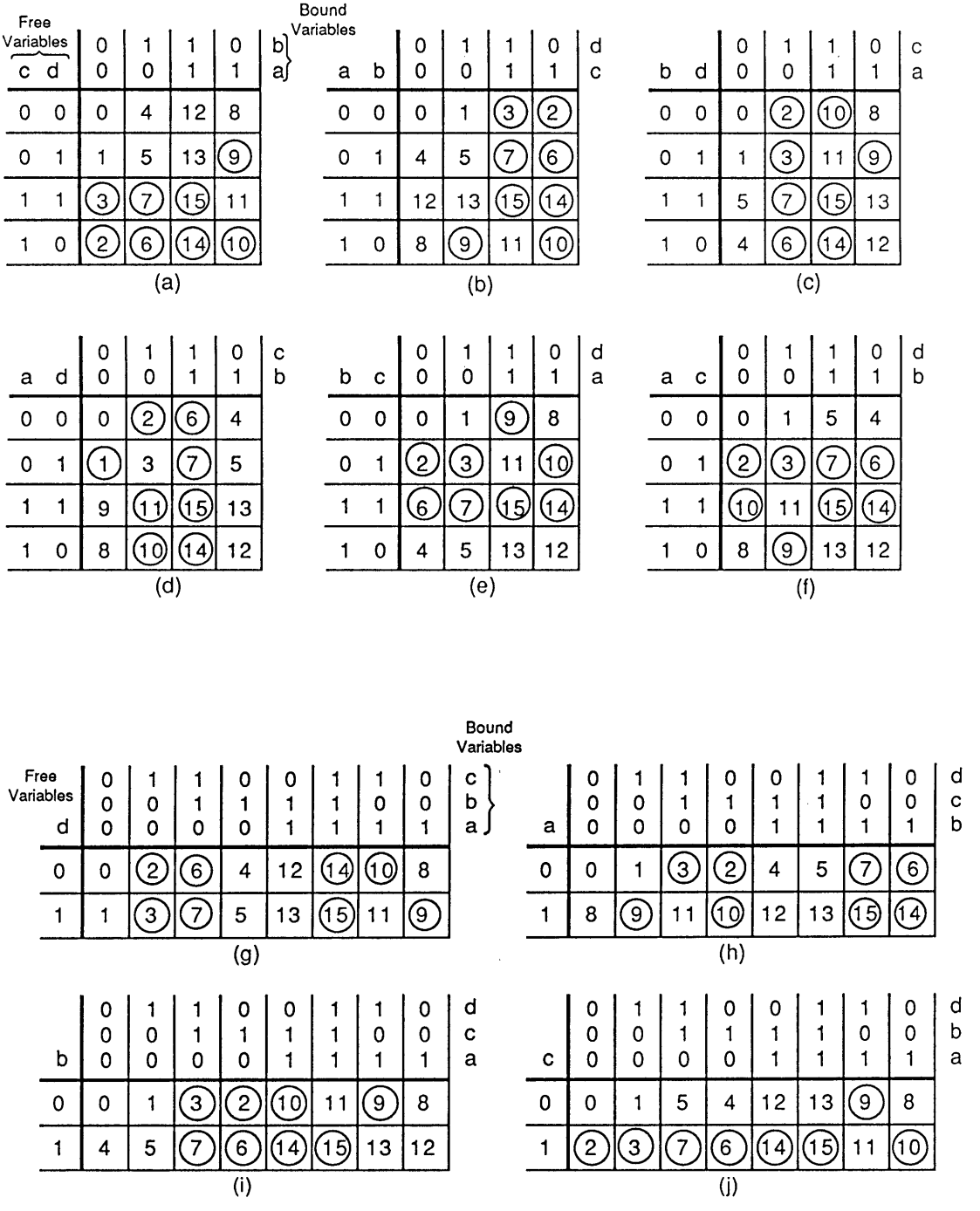


Figure 4-27 Decomposition maps for four-variable functions

For map(a)
$f(a,b,c,d)=(0)\bar{c}\bar{d}+(1)c\bar{d}+(a\bar{b})\bar{c}d+\overline{(a\bar{b})}cd=F\{g(a,b),c,d\}$ (4-34a)

For map(e)
$f(a,b,c,d)=(0)b\bar{c}+(1)bc+(ad)\bar{b}\bar{c}+\overline{(ad)}\bar{b}c=F\{g(a,d),b,c\}$ (4-34b)

For map(f)
$f(a,b,c,d)=(0)\bar{a}\bar{c}+(1)\bar{a}c+(\bar{b}d)a\bar{c}+\overline{(\bar{b}d)}ac=F\{g(b,d),a,c\}$ (4-34c)

For map(j)
$f(a,b,c,d)=\overline{(a\bar{b}d)}\bar{c}+(a\bar{b}d)c=F\{g(a,b,d),c\}$ (4-34d)

Figure 4-28 shows a NAND implementation of equation (4-34a)

Figure 4-29 shows a NAND implementation of equation (4-34d)

It should be remembered that the primary goal of decomposition is not necessarily the reduction of circuit complexity but rather to identify logical subsystems which may be more conveniently treated separately in both design and implementation or to reduce the number of gated inputs required on a single gate in the implementing logic.

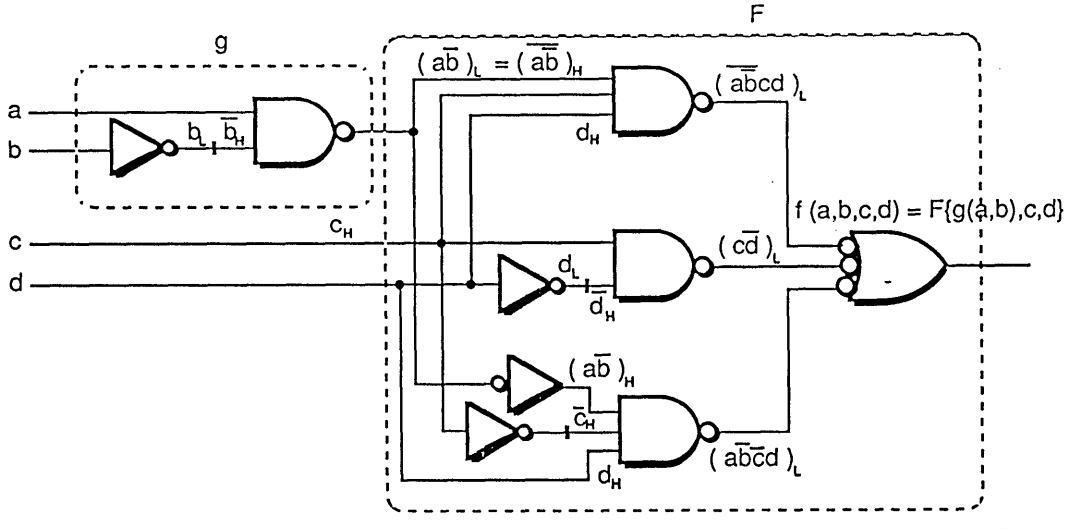


Figure 4-28 NAND implementation of $F\{g(a,b),c,d\} = a\bar{b}(\bar{c}d)+\overline{a\bar{b}}(cd)+c\bar{d}$

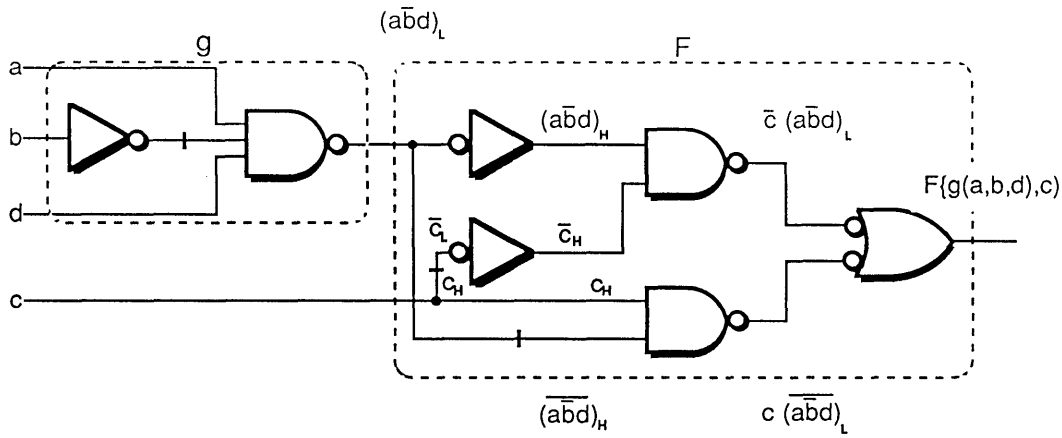


Figure 4-29 NAND implementaion of $F\{g(a,b,d),c\} = (a\bar{b}d)\,\bar{c} + \overline{(a\bar{b}d)}\,c$

There are two final comments which should be made regarding simple disjoint decomposition. (1) The previous demonstrations show that the process is quite simple if one has a prepared set of decomposition maps. Figure 4-24 gives a complete set for four variables. Figure 4-30 gives a complete set of decomposition maps for five variables. Note that some texts present chart sets in a more concise form by using a single chart for example when the bound variables are a, b and the free variables are c, d OR when the bound variables are c, d and the free variables are a, b. The argument for this is that one merely needs to rotate the chart for the alternate interpretation.

Functions containing logical redundancies can be handled on decomposition maps without difficulty. Remember that one interprets (assigns) each redundant minterm to make the column multiplicity two or less. The designer must differentiate between the circled functional minterms and the redundant minterms. As an example, consider the five-variable function

$$f(a,b,c,d,e)=\sum 2,6,7,9,12,25,27,29+X(0,8,11,13,20,22,24,28,30) \quad (4\text{-}35)$$

A possible decomposition map for this function where a, b are the free variables and c, d, e are the bound variables is shown in Fig. 4-30.

a b	000 e d c	100	110	010	001	101	111	011
0 0	~~8~~	1	3	(2)	(6)	(7)		
0 1	~~8~~	(9)	~~11~~	10			~~13~~	(12)
1 1	~~24~~	(25)	(27)	26	~~30~~		(29)	~~28~~
1 0	16	17	19	18	~~22~~			~~20~~

a b	000 e d c	100	110	010	001	101	111	011
0 0	(0)			(2)	(6)	(7)		
0 1		(9)	(11)				(13)	(12)
1 1		(25)	(27)				(29)	(28)
1 0								

Figure 4-30 Decomposition map for equation (4-35) with appropriate redundant minterm assignments

If redundant minterms 0, 11, 13, and 28 are interpreted as one, the column multiplicity becomes two and f(a,b,c,d,e) decomposes to

$$f(a,b,c,d) = b(\bar{c}d + c\bar{d}) + \bar{a}b\overline{(\bar{c}d + c\bar{d})} \quad (4\text{-}36)$$

A set of decomposition maps for five variables is shown in Fig. 4-31. Note that this figure continues over several pages. The reader is referred to the program titled DECOMPOSITION in the appendix.

(a)

a b \ e d c	000	001	010	011	100	101	110	111
0 0	0	1	3	2	6	7	5	4
0 1	8	9	11	10	14	15	13	12
1 1	24	25	27	26	30	31	29	28
1 0	16	17	19	18	22	23	21	20

(b)

a c \ e d b	000	001	010	011	100	101	110	111
0 0	0	1	3	2	10	11	9	8
0 1	4	5	7	6	14	15	13	12
1 1	20	21	23	22	30	31	29	28
1 0	16	17	19	18	26	27	25	24

(c)

b c \ e d a	000	001	010	011	100	101	110	111
0 0	0	1	3	2	18	19	17	16
0 1	4	5	7	6	22	23	21	20
1 1	12	13	15	14	30	31	29	28
1 0	8	9	11	10	26	27	25	24

(d)

c d \ e b a	000	001	010	011	100	101	110	111
0 0	0	1	9	8	24	25	17	16
0 1	2	3	11	10	26	27	19	18
1 1	6	7	15	14	30	31	23	22
1 0	4	5	13	12	28	29	21	20

(e)

b d \ e c a	000	001	010	011	100	101	110	111
0 0	0	1	5	4	20	21	17	16
0 1	2	3	7	6	22	23	19	18
1 1	10	11	15	14	30	31	27	26
1 0	8	9	13	12	28	29	25	24

(f)

a d \ e c b	000	001	010	011	100	101	110	111
0 0	0	1	5	4	12	13	9	8
0 1	2	3	7	6	14	15	11	10
1 1	18	19	23	22	30	31	27	26
1 0	16	17	21	20	28	29	25	24

(g)

c e \ d b a	000	001	010	011	100	101	110	111
0 0	0	2	10	8	24	26	18	16
0 1	1	3	11	9	25	27	19	17
1 1	5	7	15	13	29	31	23	21
1 0	4	6	14	12	28	30	22	20

(h)

b e \ d c a	000	001	010	011	100	101	110	111
0 0	0	2	6	4	20	22	18	16
0 1	1	3	7	5	21	23	19	17
1 1	9	11	15	13	29	31	27	25
1 0	8	10	14	12	28	30	26	24

(i)

a e \ d c b	000	001	010	011	100	101	110	111
0 0	0	2	6	4	12	14	10	8
0 1	1	3	7	5	13	15	11	9
1 1	17	19	23	21	29	31	27	25
1 0	16	18	22	20	28	30	26	24

(j)

d e \ c b a	000	001	010	011	100	101	110	111
0 0	0	4	12	8	20	22	20	16
0 1	1	5	13	9	25	29	21	17
1 1	3	7	15	11	27	31	23	19
1 0	2	6	14	10	26	30	22	18

Figure 4-31 Decomposition maps (a thru y) for five variables (continued)

(k)

a	b	c	0 0	1 0	1 1	0 1	e d
0	0	0	0	1	3	2	
0	0	1	4	5	7	6	
0	1	1	12	13	15	14	
0	1	0	8	9	11	10	
1	1	0	24	25	27	26	
1	1	1	28	29	31	30	
1	0	1	20	21	23	22	
1	0	0	16	17	19	18	

(l)

b	c	d	0 0	1 0	1 1	0 1	e a
0	0	0	0	1	17	16	
0	0	1	2	3	19	18	
0	1	1	6	7	23	22	
0	1	0	4	5	21	20	
1	1	0	12	13	29	28	
1	1	1	14	15	31	30	
1	0	1	10	11	27	26	
1	0	0	8	9	25	24	

(m)

a	c	d	0 0	1 0	1 1	0 1	e b
0	0	0	0	1	9	8	
0	0	1	2	3	11	10	
0	1	1	6	7	15	14	
0	1	0	4	5	13	12	
1	1	0	20	21	29	28	
1	1	1	22	23	31	30	
1	0	1	18	19	27	26	
1	0	0	16	17	25	24	

(n)

a	b	d	0 0	1 0	1 1	0 1	e c
0	0	0	0	1	5	4	
0	0	1	2	3	7	6	
0	1	1	10	11	15	14	
0	1	0	8	9	13	12	
1	1	0	24	25	29	28	
1	1	1	26	27	31	30	
1	0	1	18	19	23	22	
1	0	0	16	17	21	20	

(o)

b	c	e	0 0	1 0	1 1	0 1	d a
0	0	0	0	2	18	16	
0	0	1	1	3	19	17	
0	1	1	5	7	23	21	
0	1	0	4	6	22	20	
1	1	0	12	14	30	28	
1	1	1	13	15	31	29	
1	0	1	9	11	27	25	
1	0	0	8	10	26	24	

(p)

a	c	e	0 0	1 0	1 1	0 1	d b
0	0	0	0	2	10	8	
0	0	1	1	3	11	9	
0	1	1	5	7	15	13	
0	1	0	4	6	14	12	
1	1	0	20	22	30	28	
1	1	1	21	23	31	29	
1	0	1	17	19	27	25	
1	0	0	16	18	26	24	

Figure 4-31 Decomposition maps (a thru y) for five variables (continued)

(q)

a	b	e	0 0	1 0	1 1	0 1	d c
0	0	0	0	2	6	4	
0	0	1	1	3	7	5	
0	1	1	9	11	15	13	
0	1	0	8	10	14	12	
1	1	0	24	26	30	28	
1	1	1	25	27	31	29	
1	0	1	17	19	23	21	
1	0	0	16	18	22	20	

(r)

a	d	e	0 0	1 0	1 1	0 1	c b
0	0	0	0	4	12	8	
0	0	1	1	5	13	9	
0	1	1	3	7	15	11	
0	1	0	2	6	14	10	
1	1	0	18	22	30	26	
1	1	1	19	23	31	27	
1	0	1	17	21	29	25	
1	0	0	16	20	28	24	

(s)

b	d	e	0 0	1 0	1 1	0 1	c a
0	0	0	0	4	20	16	
0	0	1	1	5	21	17	
0	1	1	3	7	23	19	
0	1	0	2	6	22	18	
1	1	0	10	14	30	26	
1	1	1	11	15	31	27	
1	0	1	9	13	29	25	
1	0	0	8	12	28	24	

(t)

c	d	e	0 0	1 0	1 1	0 1	b a
0	0	0	0	8	24	16	
0	0	1	1	9	25	17	
0	1	1	3	11	27	19	
0	1	0	2	10	26	18	
1	1	0	6	14	30	22	
1	1	1	7	15	31	23	
1	0	1	5	13	29	21	
1	0	0	4	12	28	20	

(u)

a	0 0 0 0	1 0 0 0	1 1 0 0	0 1 0 0	0 1 1 0	1 1 1 0	1 0 1 0	0 0 1 0	0 0 1 1	1 0 1 1	1 1 1 1	0 1 1 1	0 1 0 1	1 1 0 1	1 0 0 1	0 0 0 1	e d c b
0	0	1	3	2	6	7	5	4	12	13	15	14	10	11	9	8	
1	16	17	19	18	22	23	21	20	28	29	31	30	26	27	25	24	

(v)

b	0 0 0 0	1 0 0 0	1 1 0 0	0 1 0 0	0 1 1 0	1 1 1 0	1 0 1 0	0 0 1 0	0 0 1 1	1 0 1 1	1 1 1 1	0 1 1 1	0 1 0 1	1 1 0 1	1 0 0 1	0 0 0 1	e d c a
0	0	1	3	2	6	7	5	4	20	21	23	22	18	19	17	16	
1	8	9	11	10	14	15	13	12	28	29	31	30	26	27	25	24	

Figure 4-31 Decomposition maps (a thru y) for five variables (continued)

(w)	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	e
	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	d
	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	b
c	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	a
0	0	1	3	2	10	11	9	8	24	25	27	26	18	19	17	16	
1	4	5	7	6	14	15	13	12	28	29	31	30	22	23	21	20	

(x)	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	e
	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	c
	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	b
d	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	a
0	0	1	5	4	12	13	9	8	24	25	29	28	20	21	17	16	
1	2	3	7	6	14	15	11	10	26	27	31	30	22	23	19	18	

(y)	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	d
	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	c
	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	b
e	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	a
0	0	2	6	4	12	14	10	8	24	26	30	28	20	22	18	16	
1	1	3	7	5	13	15	11	9	25	27	31	29	21	23	19	17	

Figure 4-31 Decomposition maps (a thru y) for five variables

ANALYSIS OF MULTILEVEL COMBINATIONAL LOGIC

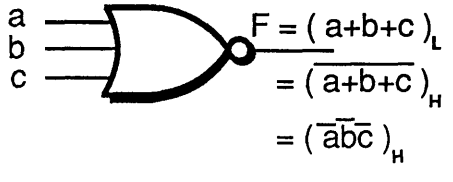
During the decade of the 1950's great strides were made in the development of algorithms and other design aids to assist in the simplification, analysis, and synthesis of combinational logical circuitry. Although the main thrust of this text is synthesis, we must also deal with the problems of analysis of both combinational and sequential networks. Here we will deal with the analysis of the combinational circuit.

We have already learned how to interpret a properly constructed 806 B logic diagram and this interpretation is a first step in the analysis procedure. The interpretation, however, falls short of dealing with the problems of static and dynamic hazards in a network. This real problem exists but is not accounted for in our usual algebraic or logic diagram descriptions. Three works by Maley and Earle[3], Huffman[4], and Erb[5] permit the complete analysis of the problem from the standpoint of both logic and hazard analysis. The material to follow is based on components of these three references with the author's departure to make the collective presentation in conformity with the notation and style of the overall text.

Combinational Analysis Using K-Maps at the Gate Level

We will make the practical assumption that the gating structures of our combinational circuits will be in either NOR or NAND form. Our task will be to develop algorithms to permit the construction and interpretation of logic diagrams in which each NAND or NOR gate is replaced by a K-map to permit the rapid analysis of the network's output as a function of the input variables. Then, from this K-map logic diagram we will learn how to identify possible network hazards and how to correct them to make the network hazard-free. The method presented here follows Erb (reference 5).

Plotting K-maps for NOR networks - A NOR gate operating as a logic OR has the symbol and Karnaugh map shown in Fig. 4-32(a), (b) and (c) respectively.



(a)

a b \ c	0	1
0 0	1	0
0 1	0	0
1 1	0	0
1 0	0	0

(b)

a b \ d c	0 0	1 0	1 1	0 1
0 0	1	1	0	0
0 1	0	0	0	0
1 1	0	0	0	0
1 0	0	0	0	0

(c)

Figure 4-32 Symbol and Karnaugh maps of a NOR gate

The output of a NOR gate is depicted on the Karnaugh map as the "loop" containing the Boolean product (b) of its input variables or the "loop" containing the reduced Boolean product (c) of its input variables.

For a two-level NOR circuit as shown in Fig. 4-33 the

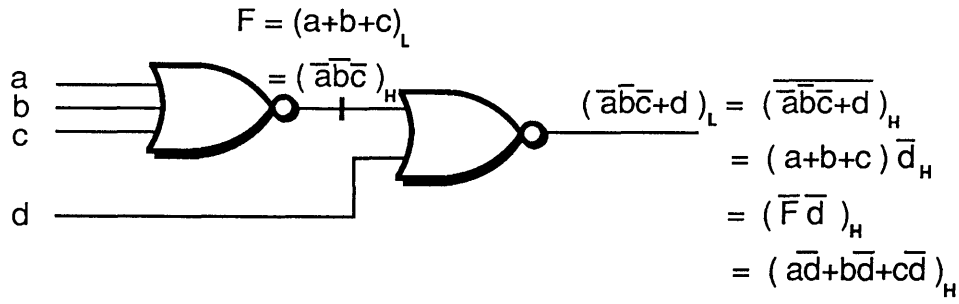


Figure 4-33 Two-level NOR network

Karnaugh map development of the output function is shown in Fig. 4-34. Here the loop for $\bar{a}\bar{b}\bar{c}$ in the $\bar{F}$ map cancelled

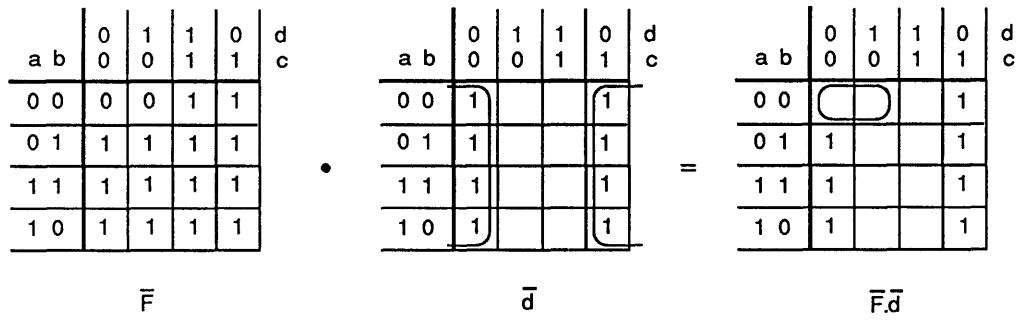


Figure 4-34 Karnaugh map development of the output of figure 4-33

the $\bar{a}\bar{b}\bar{c}$ minterm in the $\bar{d}$ map, leaving $a\bar{d} + b\bar{d} + c\bar{d}$.

From this little demonstration we may write our first NOR gate algorithm as

NOR 1. For a NOR gate, the loop (here $\bar{a}\bar{b}\bar{c}$) of a preceding gate inhibits those parts of the loop (here $\bar{d}$) generated by the input variable(s) of the following gate.[5]

The Karnaugh map at the gate level may be drawn as shown in Fig. 4-35. NOTE - d is plotted on the map as $\bar{d}$ just as abc

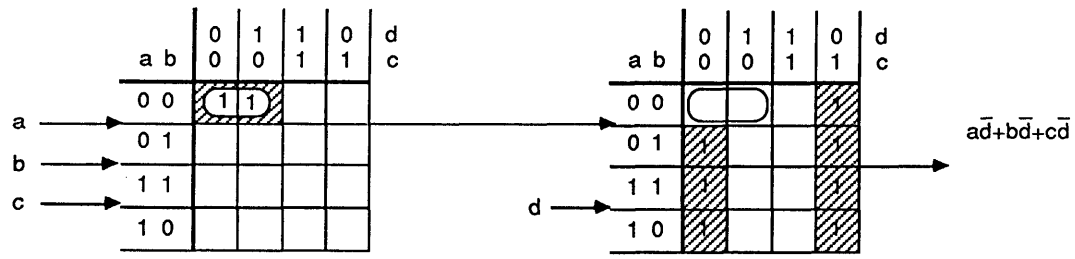


Figure 4-35 Algorithmic plot of figure 4-32

is plotted as $\bar{a}\bar{b}\bar{c}$, as shown in Fig. 4-32.

There is a second condition which must be addressed which will lead to our second algorithm, NOR 2. This is the condition where no input variables go directly to a NOR gate input, i.e. the NOR gate's inputs are derived from other NOR gate outputs. Here our algorithm NOR 2 is

NOR 2. For a NOR gate which has no system input variable going directly as an input to the gate, its entire map will be filled with 1's except those minterms which are inhibited by the outputs of preceding gates.[5]

For example, consider the two-level network of Fig. 4-36.

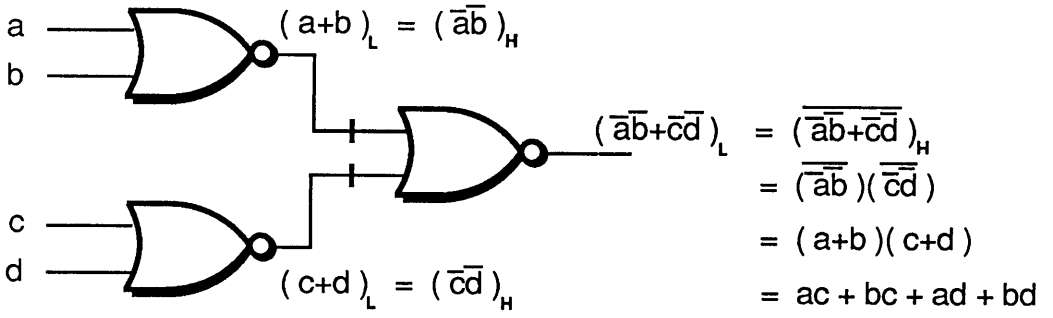


Figure 4-36 Two-level NOR/NOR network

The gate level Karnaugh mapping of Fig. 4-36 is shown in Fig. 4-37.

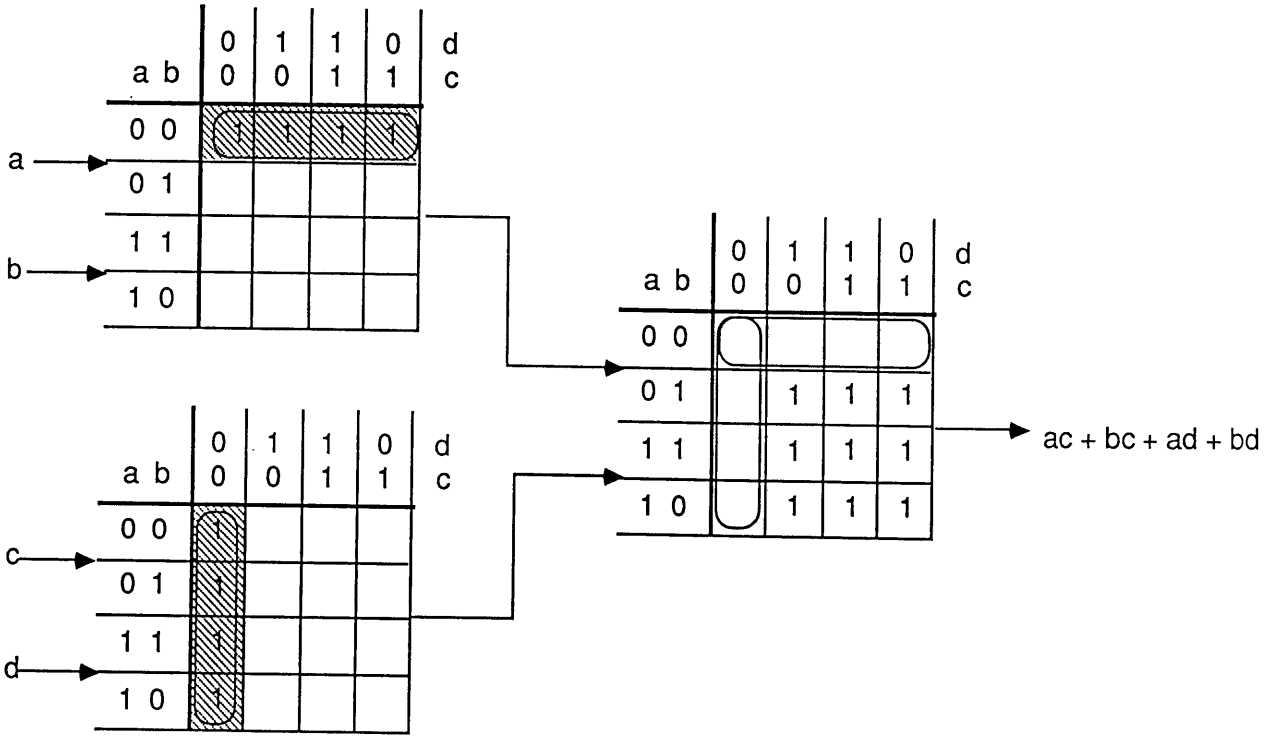


Figure 4-37 Gate-level Karnaugh mapping of the two-level NOR/NOR network of figure 4-36

Plotting K-maps for NAND networks - A symmetrical treatment may be made for the NAND gate. A NAND gate operating as a logic AND has the symbol and Karnaugh maps shown in Fig. 4-38 (a), (b), and (c) respectively.

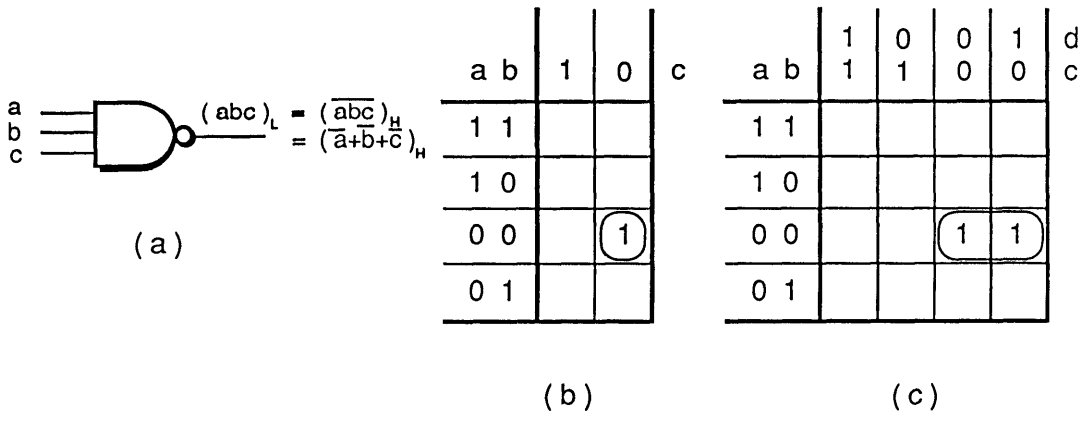


Figure 4-38 Symbol and Karnaugh maps of a NAND gate

The output of a NAND gate is depicted on the Maxterm Karnaugh map as the loop containing the Boolean sum (b) of its input variables or the loop containing the reduced Boolean sum (c) of its input variables.

The algorithms for NAND logic interpretation when Karnaugh mapped at the gate level are

NAND 1. For a NAND gate, the loop generated by a preceding NAND gate inhibits those parts of the loops generated by the input variable(s) of the following gate.

NAND 2. For a NAND gate which has no system input variable going directly as an input to the gate, its entire map (Maxterm) will be filled with 1's except those Maxterms which are inhibited by outputs of preceding gates.

For the two-level NAND network shown in Fig. 4-39 the Karnaugh map at the gate level is shown in Fig. 4-40. Note, again, that these maps are Maxterm maps.

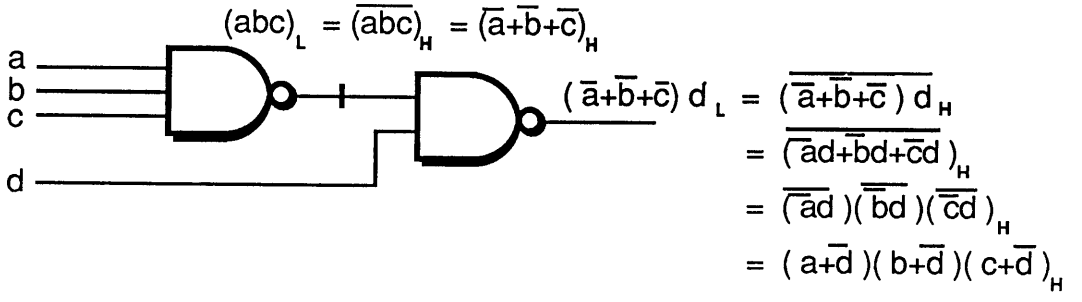


Figure 4-39 Two-level NAND gate network

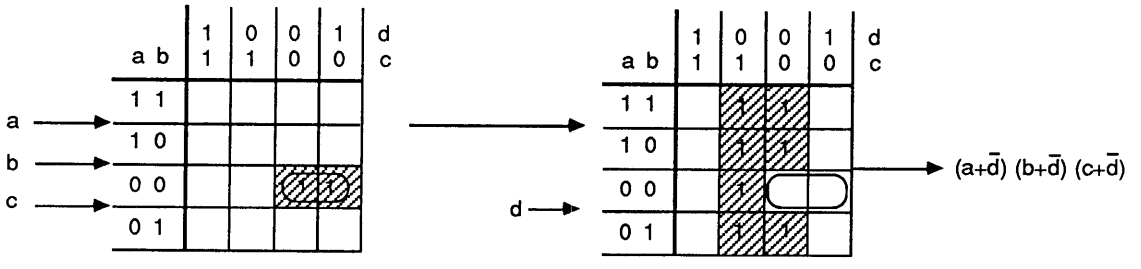


Figure 4-40 Algorithmic plot of figure 4-39

Now consider the two-level NAND network shown in Fig. 4-41.

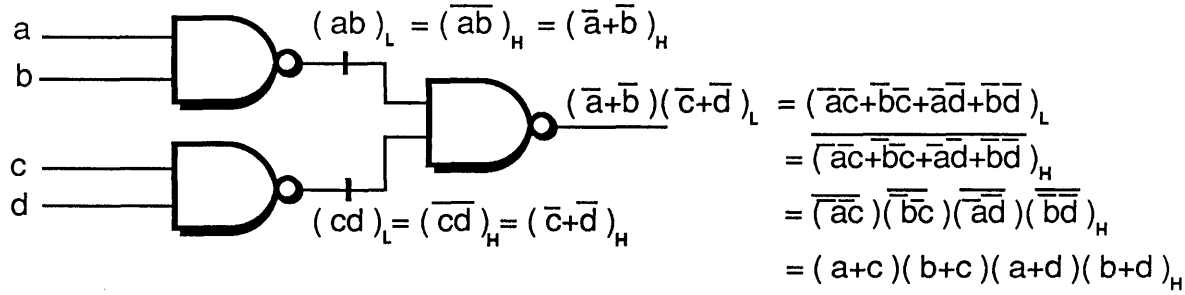


Figure 4-41 Two-level NAND/NAND gate network

The gate level Karnaugh mapping of Fig. 4-41 is shown in Fig. 4-42.

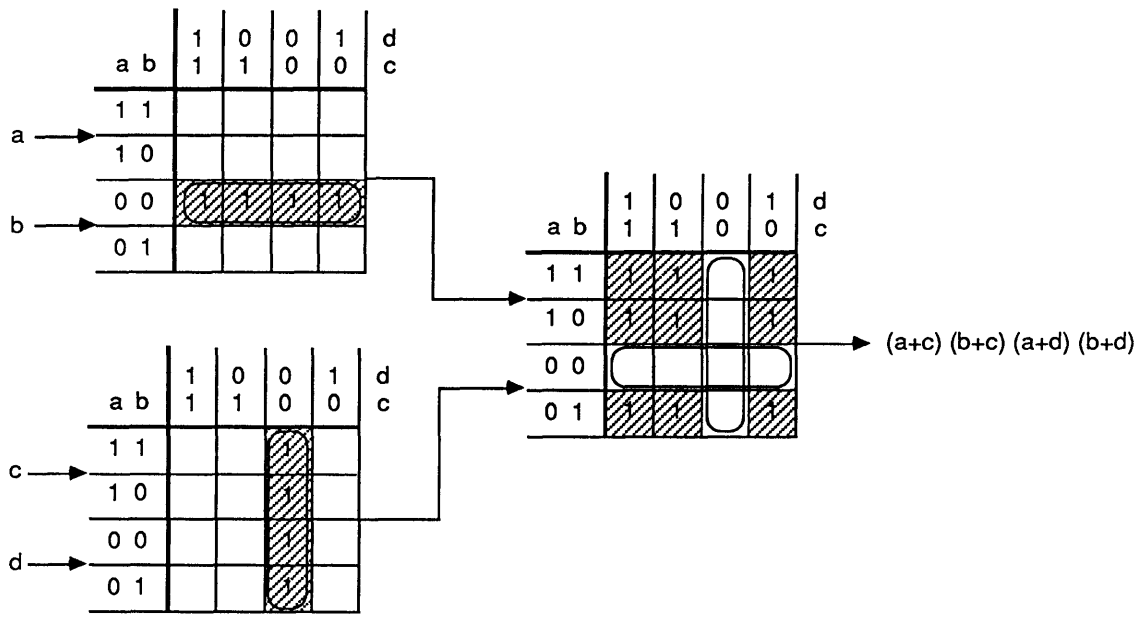


Figure 4-42 Gate-level Karnaugh mapping of the two-level NAND/NAND network of figure 4-41

Now that the basic ideas of utilizing the Karnaugh maps at the gate level to analyze logic diagrams is understood, let

us analyze the logic network of Fig. 4-43. This network implements the pair of logical equations for a serial binary adder as derived in Chapter I.

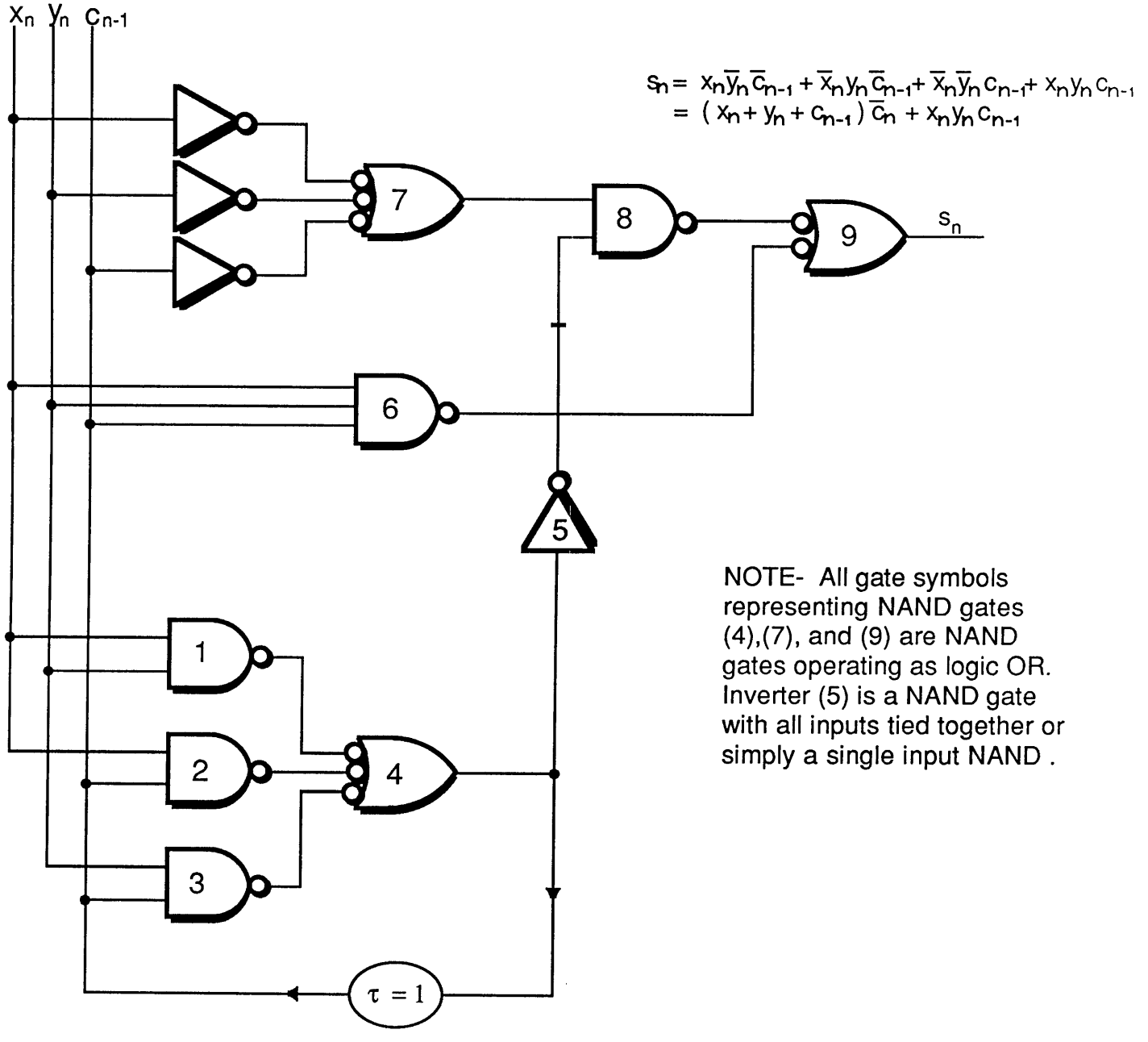


Figure 4-43 NAND logic implementation of the sum equation for a serial binary adder

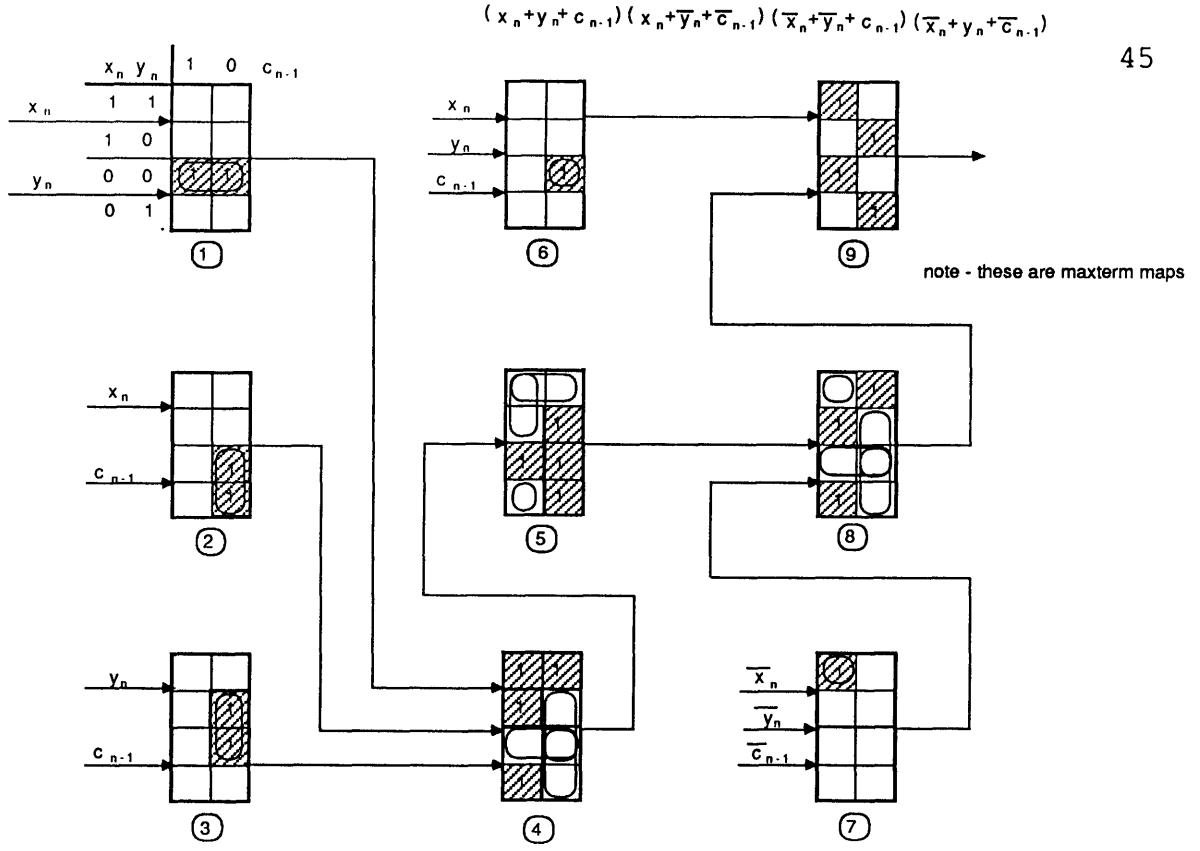


Figure 4-44 Gate-level Karnaugh map analysis of figure 4-43

Comparing the output of Figs. 4-43 for s_n with the Karnaugh map analysis at the gate level of Fig. 4-44 confirms the correctness of the logical network.

As a final example we will analyze the NOR circuit of Fig. 4-45 using gate-level Karnaugh map analysis. This map analysis is shown in Fig. 4-46. Note that the maps of Fig. 4-46 are minterm maps.

Gate-level Karnaugh mapping may also be used as a synthesis tool whereby one produces a map of the desired output and works backward toward the system inputs to develop the necessary network logic.

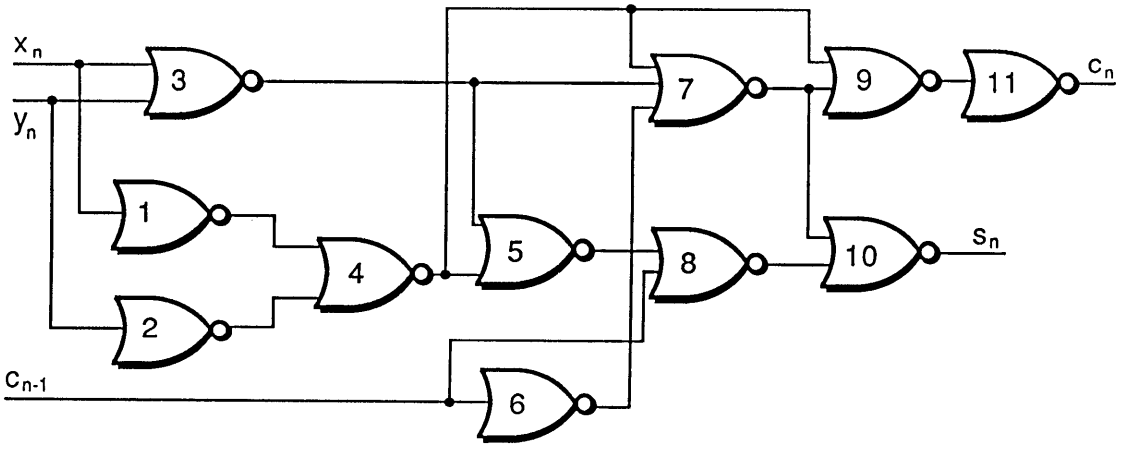


Figure 4-45 A pictorial diagram for the interconnection of eleven NOR gates to generate the sum and carry equations of a serial binary adder. Note that this diagram does not necessarily conform to the rules given for a logic diagram to 806 B standards

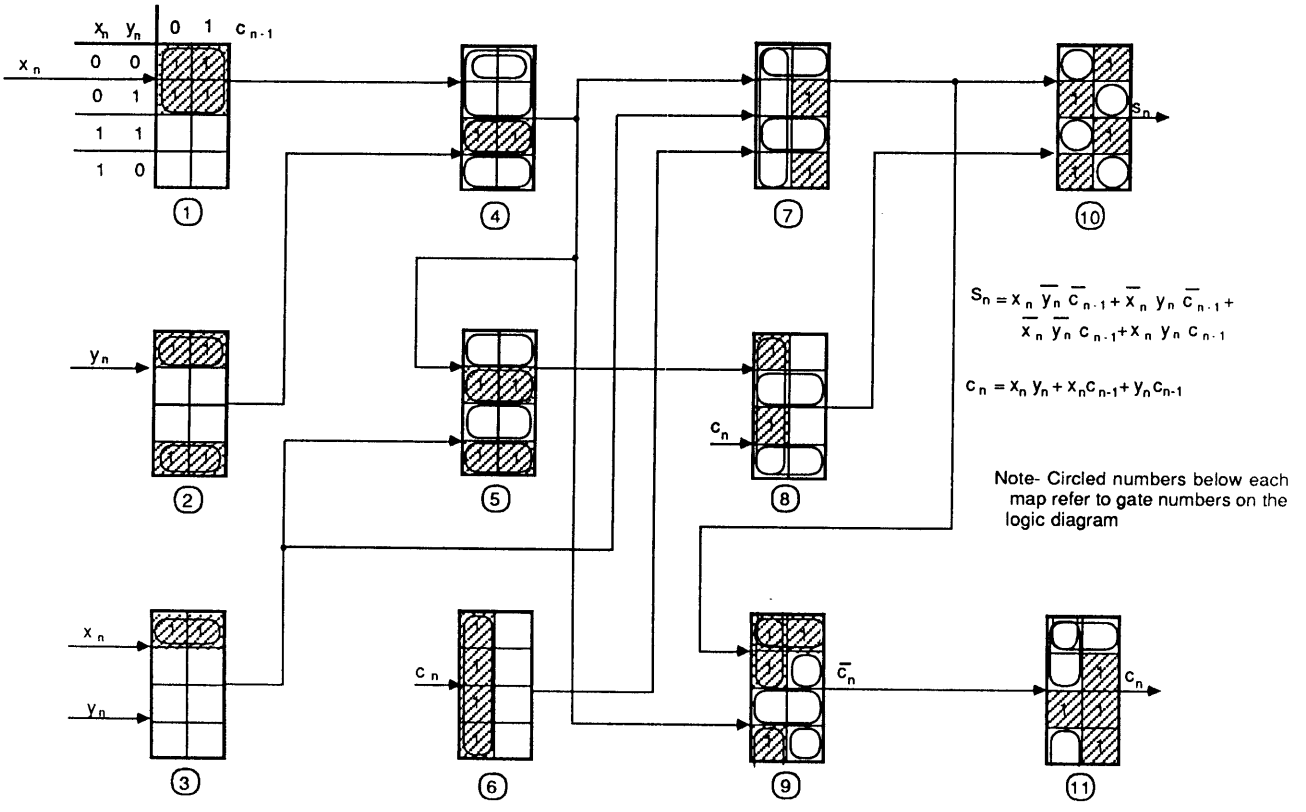


Figure 4-46 Gate-level Karnaugh map analysis of figure 4-45

The Analysis of Static Hazards

The equations we derived for combinational networks make the major assumption that the operating time of a gate is zero, i.e. that at any instant of time n, the gate's output is precisely the indicated logical function of its inputs. This assumption regarding the output is true in the steady-state but it is not necessarily true during the gate delay time-interval following an input change. Let us look at common TTL two-input NAND and NOR gates and relate their outputs to their inputs as shown in Fig. 4-47.

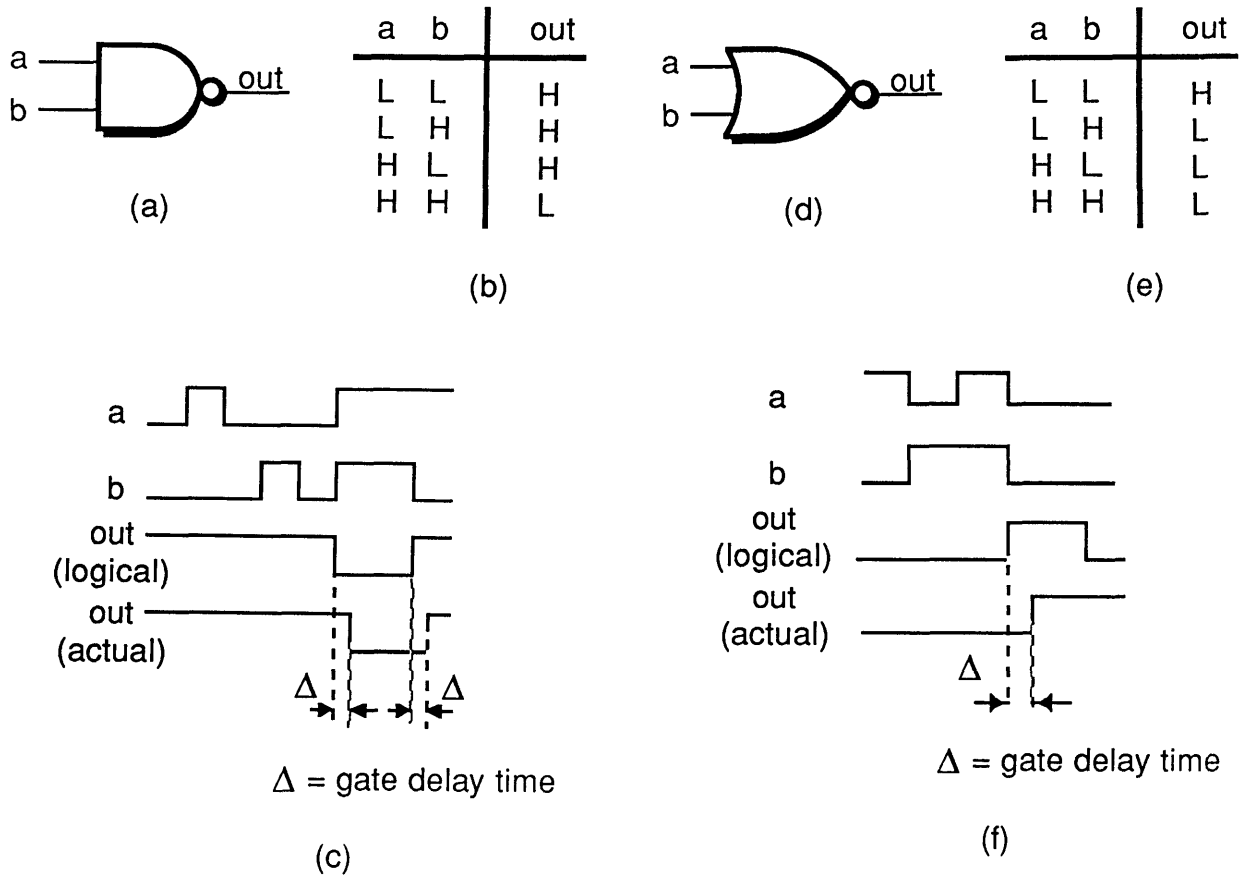


Figure 4-47 (a) NAND gate
(b) Voltage table (steady state)
(c) Actual response
(d) NOR gate
(e) Voltage table (steady state)
(f) Actual response

This indicated propagation delay time Δ is of the order of 10ns in a typical TTL gate. (Either regular or low-power or Schottky) and as long as 33ns for low-power TTL gates. Let us now see how these propagation delays can create a static hazard in NAND and NOR networks. Here we will define a static hazard as a condition where a change in a single system's variable produces a momentary output change not in conformity with the network's logical equation. Consider the logic of Fig. 4-48.

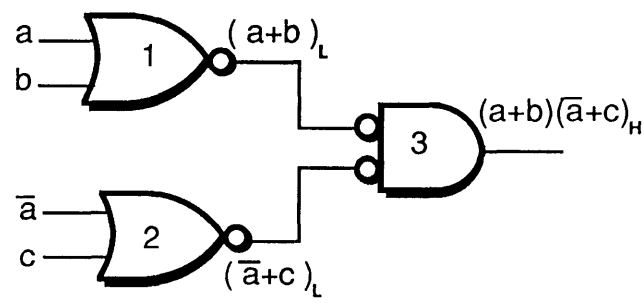


Figure 4-48 NOR gate network with a static hazard

Let us examine the time response of this network to an input combination change of the variables from a = 0, b = 0, and c = 0 to a = 1, b = 0, c = 0. We note such a change as 000 → 100. The timing diagram for this change is shown in Fig. 4-49.

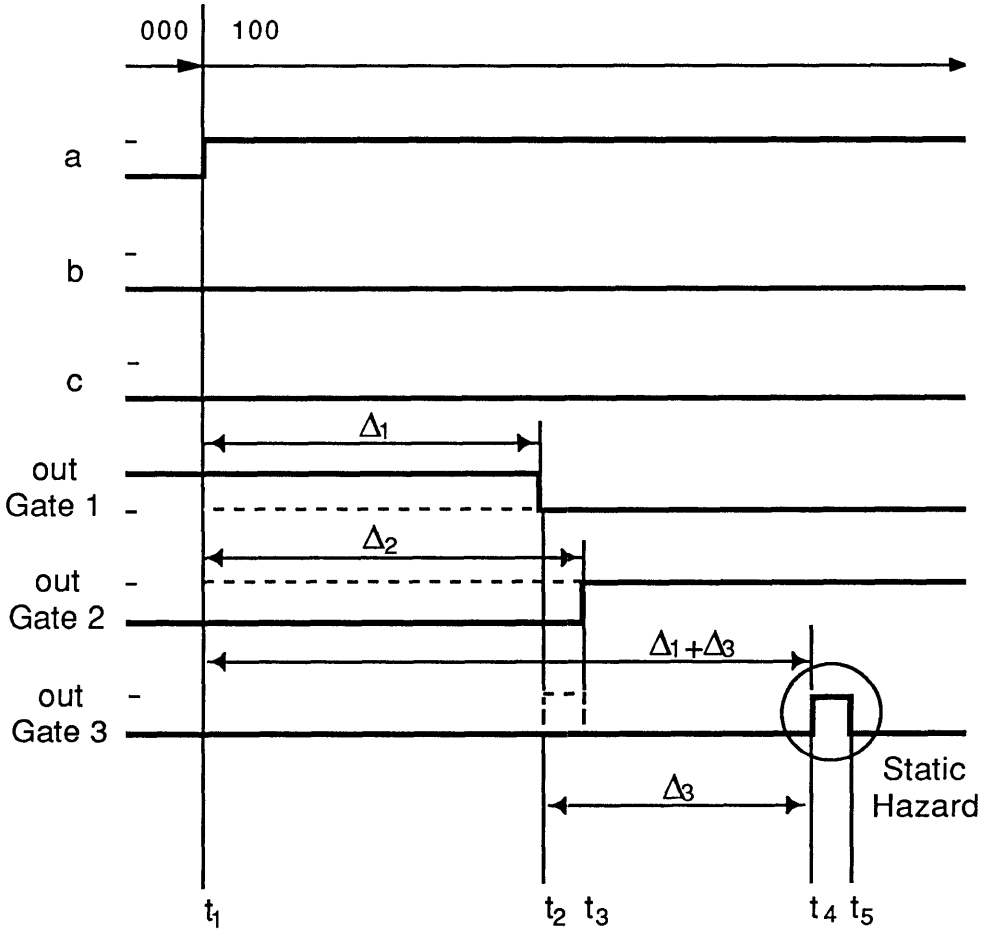


Figure 4-49 A possible timing diagram for the logic of figure 4-45 assuming $\Delta_1 < \Delta_2$

This particular timing diagram is constructed on the assumption that the gate delays Δ_1, Δ_2, and Δ_3 are unequal and further that $\Delta_1 < \Delta_2$. Here a 0 → 1 transition in a at t_1 causes a 1 → 0 transition in the output of gate #1 at t_2 and a 0 → 1 transition in the output of gate #2 at t_3. Because the gate delays are different ($\Delta_1 < \Delta_2$), gate #1 goes to 0 before gate #2 goes to 1. The 1 → 0 transition of gate #1 causes a 0 → 1 transition of gate #3 at t_4 and the 0 → 1 transition of gate #2 causes a subsequent 1 → 0

transition of gate #3 at t_5. This 0 → 1 → 0 transition of gate #3's output is a static hazard. Note that if $\Delta_1 > \Delta_2$, no hazard would be generated for a 000 → 100 change of system variables. Further analysis would show that

1. no static hazard would occur for the transition 100 → 000 (when $\Delta_2 > \Delta_1$).
2. no static hazard would occur for the transition 000 → 100 (when $\Delta_1 > \Delta_2$).
3. a static hazard would occur for the transition 100 → 000 (when $\Delta_1 > \Delta_2$).

Therefore, to exhaustively examine all input variable changes between adjacent mapped minterms with all relative values of system gate delays makes a timing diagram determination of possible hazards quite impossible. We do however have an excellent alternative means to accomplish this task.

Let us return to the gate-level Karnaugh map representation of NOR the network in Fig. 4-48 as shown in Fig. 4-50.

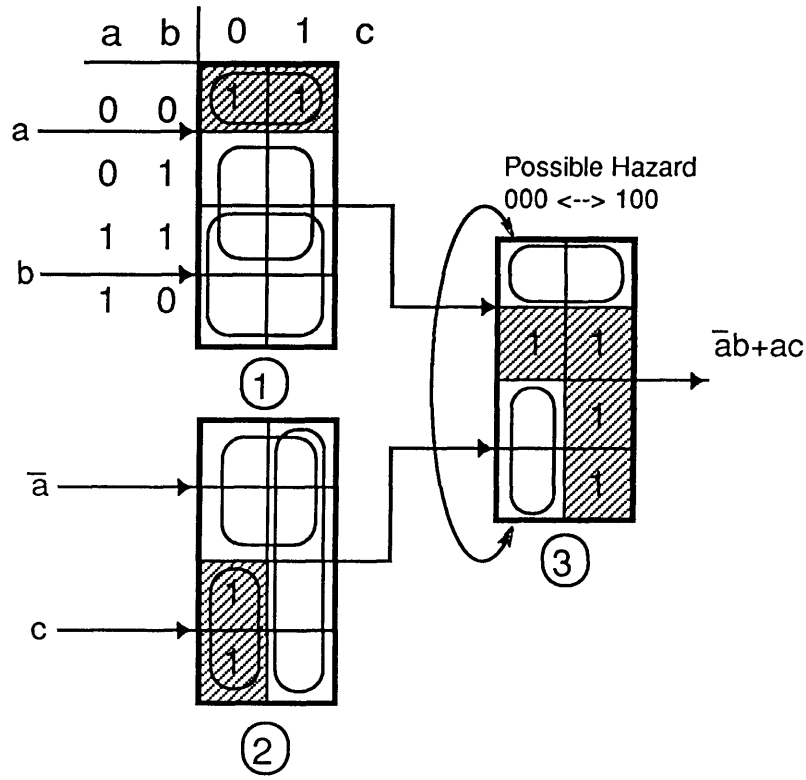


Figure 4-50 Gate-level Karnaugh map of figure 4-48

Here we can immediately recognize the transition between input variable combinations which might produce a hazard. In this case it occurs between the combinations 000 and 100 since the output function is $\bar{a}b + ac$ only if the inhibiting signals from gate #1 and gate #2 are precisely equal. If there is a difference in the time delays, the output function will momentarily be a + b if $\Delta_1 < \Delta_2$ and $\bar{a} + c$ if $\Delta_1 > \Delta_2$. This momentary change in the output signal has a duration equal to the difference in the gate delays $(\Delta_2 - \Delta_1)$ and occurs at a time $(\Delta_2 - \Delta_1)$ after the change in variable a at the input to gates #1 and #2. For the input transitions 000 → 100 the output function in transient will

be $0_{SS} \rightarrow 1 \rightarrow 0_{SS}$ for the case $\Delta_1 < \Delta_2$ and $0_{SS} \rightarrow 0 \rightarrow 0_{SS}$ for the case $\Delta_1 > \Delta_2$. We must therefore provide an additional input to gate #3 to provide an inhibiting loop between the inhibiting loops from gates #1 and #2. Specifically, we need a term ($\bar{b}\bar{c}$). This is shown in Fig. 4-51.

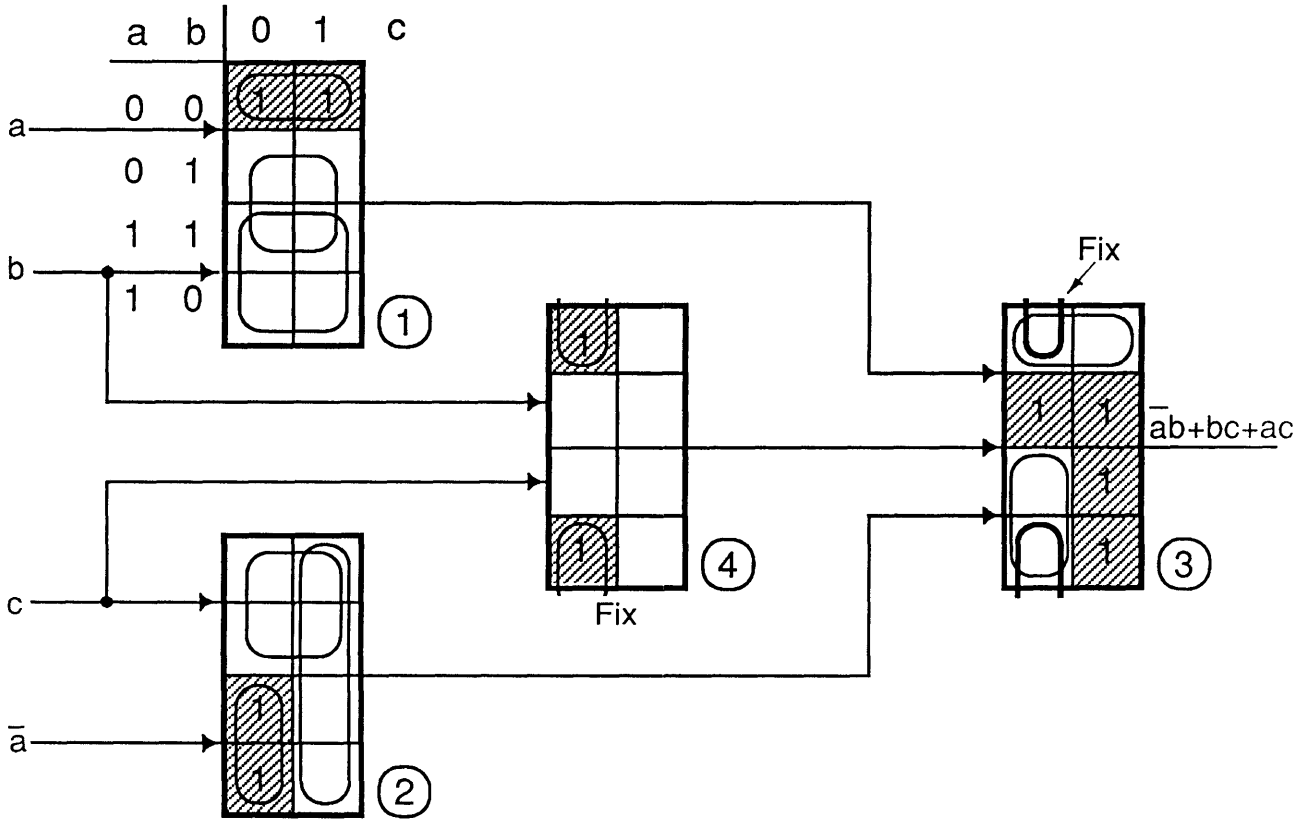


Figure 4-51 Elimination of the static hazard in gate #3 by the addition of gate #4

The addition of a logical term (bc) eliminates the hazard created by the difference in the delays of gates #1 and #2 by holding the output of gate #3 low during the interval $\Delta_2 - \Delta_1$. The generation of the (bc) term consequently adds another gate to the logic of Fig. 4-48 as shown in Fig. 4-52.

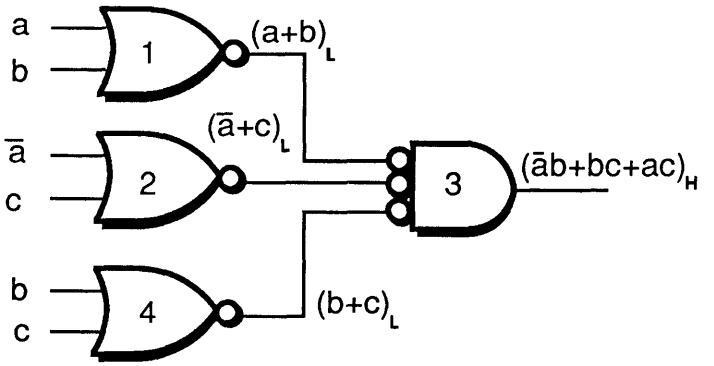


Figure 4-52 Modified circuit of figure 4-48
to form a hazard-free network

Verification that the hazard is removed can be seen in the timing diagram for Fig. 4-52 as shown in Fig. 4-53.

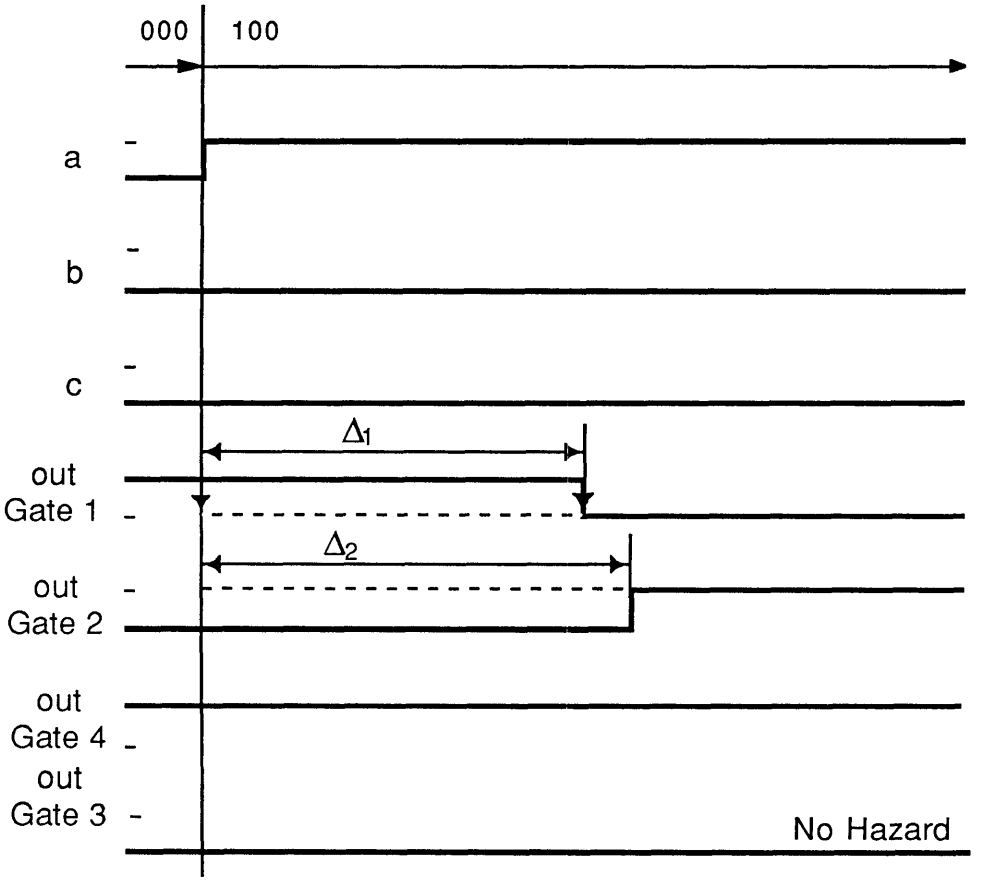


Figure 4-53 Timing diagram of the hazardless
circuit of figure 4-52
for a 000 --> 100 transition

There are two important messages in this discussion. (1) That gate-level Karnaugh maps may be interpreted to determine the existence of <u>possible</u> static hazards and (2) By the addition of other logic terms in the switching function we may eliminate the hazard and such additions are immediately identifiable on the gate-level Karnaugh maps.

If one adheres to the convention of drawing NOR gates as minterm maps and NAND gates as Maxterm maps a single set of three interpretive rules for identifying possible static hazards may be invoked. Potential static hazards are identified when

(1) a change of a system variable causes a change from one inhibiting loop to another (as in the previous example).
(2) an existing potential static hazard between 0's propagates to the next level and becomes a potential hazard between 1's.
(3) a potential hazard between 1's propagates to the next gate as a hazard between 0's.

Four examples will now be shown to illustrate the use of each rule.

<u>Example 1</u> - Consider the two-level NOR network of Fig. 4-54. This is the same example shown in Fig. 4-48. Here we <u>directly</u> identify a possible state hazard by interpreting

the gate-level Karnaugh map circuit representation shown in Fig. 4-55 according to the previously mentioned rules. Here, a potential hazard is identified for the transition 000 ↔ 100 via Rule 1.

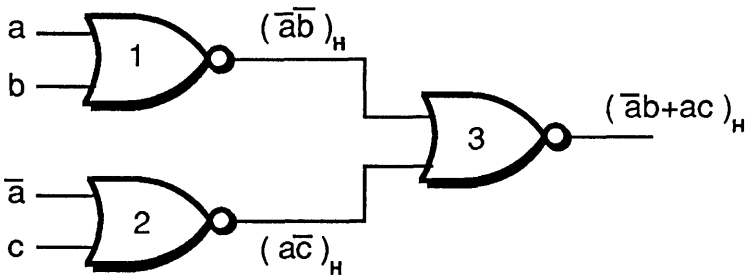


Figure 4-54 A two-level NOR network with a rule (1) hazard

The timing diagram is shown only for confirmation. Here, if Δ_1 is assumed to be greater than Δ_2 a static hazard is confirmed for the transition 100 → 000. Recall that in the previous example's timing diagram (Fig. 4-49) we assumed that $\Delta_1 < \Delta_2$ and showed the static hazard occurred on the 000 → 100 transition. Thus the reader should conclude that Rule 1 applied to gate #3 only identifies a <u>possible</u> hazard in a transition 000 ↔ 100 (i.e. 000 → 100 <u>or</u> 100 → 000). A more specific definition of the hazard is dependent on the relative delays of the gates (i.e. Δ_1, Δ_2 and Δ_3). It is, however, unnecessary to explore various relative gate delays but rather to remove <u>all</u> potential hazards, regardless of differences in the Δ's by logical modifications (as shown in Fig. 4-51, 52, and 53).

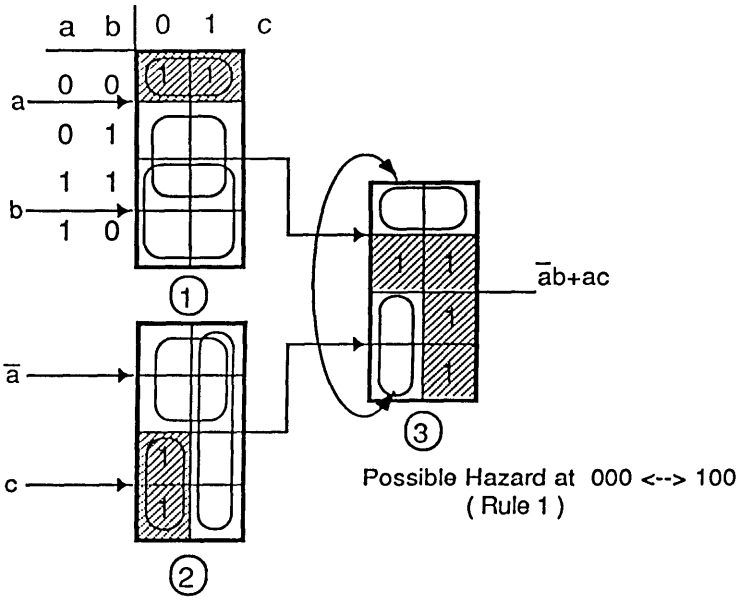


Figure 4-55 (a) Gate-level Karnaugh maps of figure 4-54

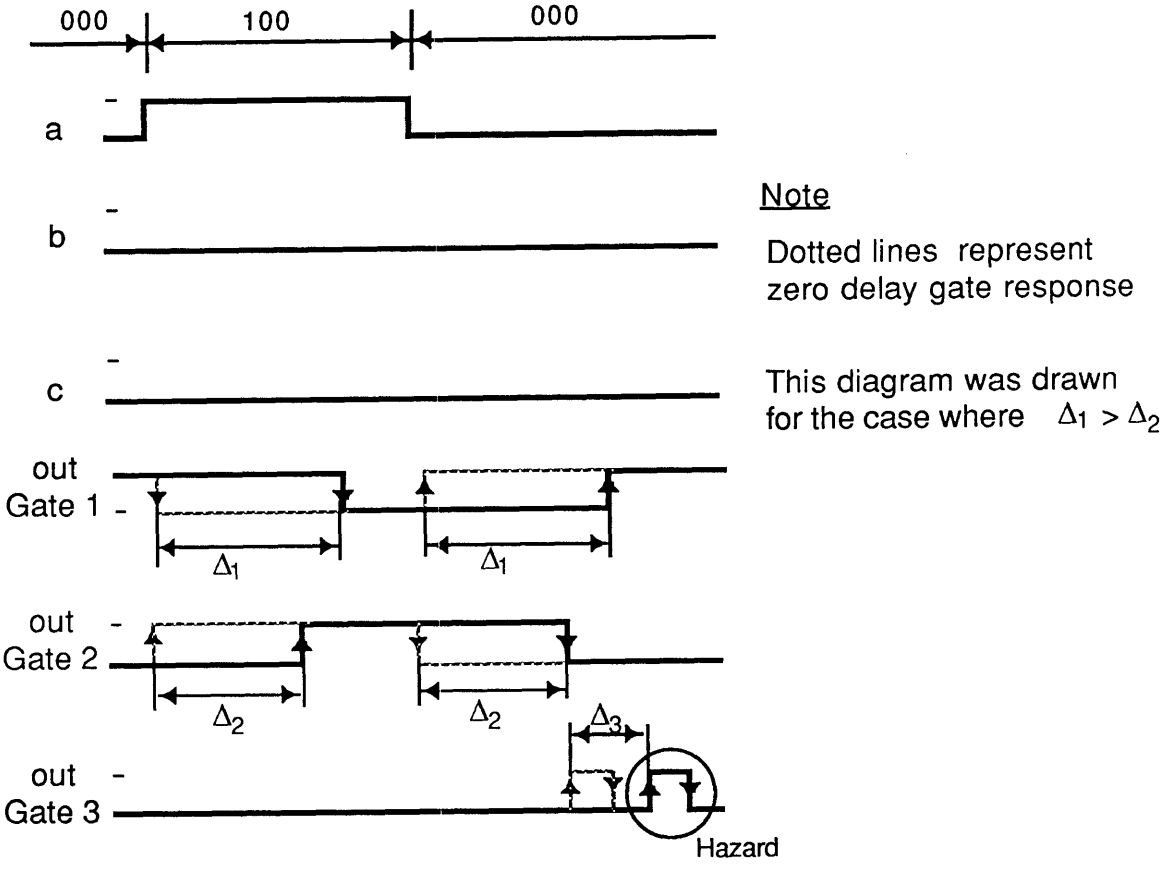


Figure 4-55 (b) Timing diagram showing hazard resulting from 100 --> 000 transition

Example 2 - Consider the three-level NOR network of Fig. 4-56.

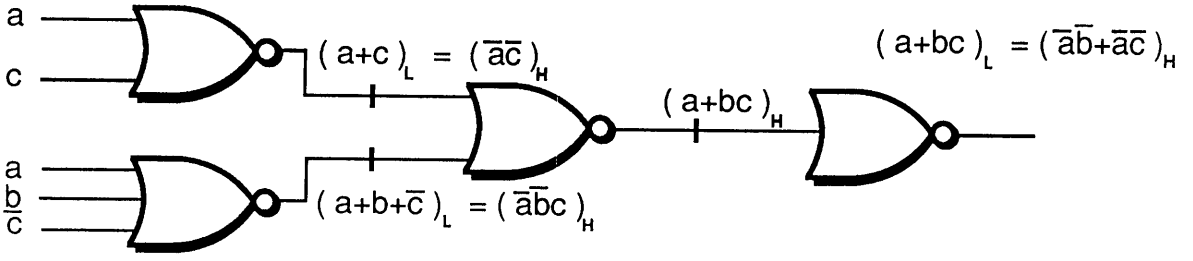


Figure 4-56 A three-level NOR network with a rule (2) hazard

The gate-level Karnaugh maps for Fig. 4-56 may be readily drawn as shown in Fig. 4-57(a). Here we determine a possible hazard in gate 3 (rule 1) and in gate 4 (rule 2).

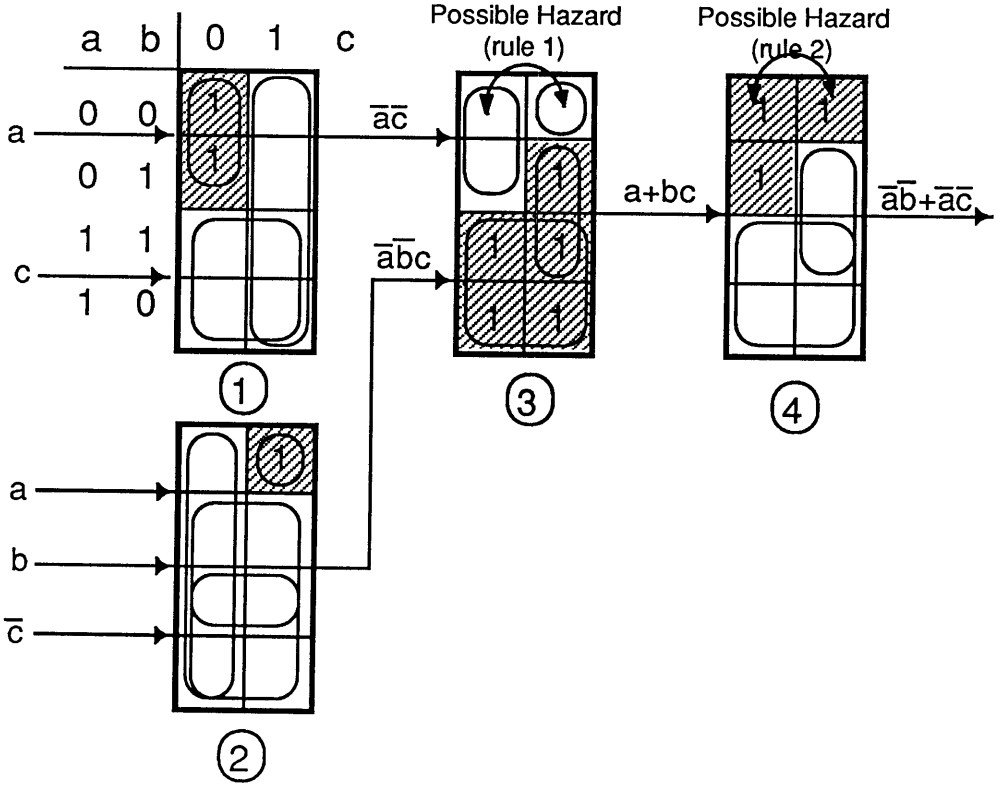


Figure 4-57(a) Gate-level Karnaugh maps for figure 5-56

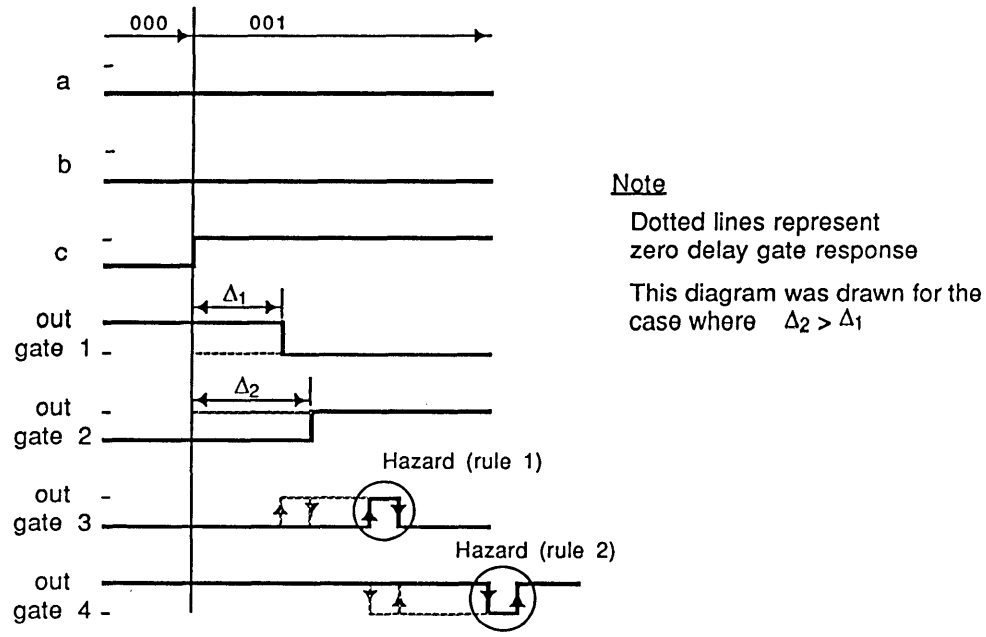


Figure 4-57(b) Timing diagram showing hazard resulting from a 000 --> 001 transition

Example 3 - Consider the four-level NOR network of Fig. 4-58.

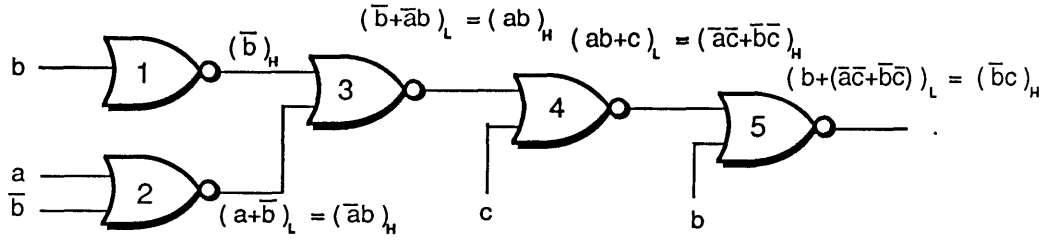


Figure 4-58 A four-level NOR network with a rule (3) hazard

The gate-level Karnaugh maps for Fig. 4-58 may be readily drawn as shown in Fig. 4-59(a). Here we find a possible Rule (1) hazard in gate 3, a possible Rule (2) hazard in gate 4, and a possible Rule (3) hazard in gate 5.

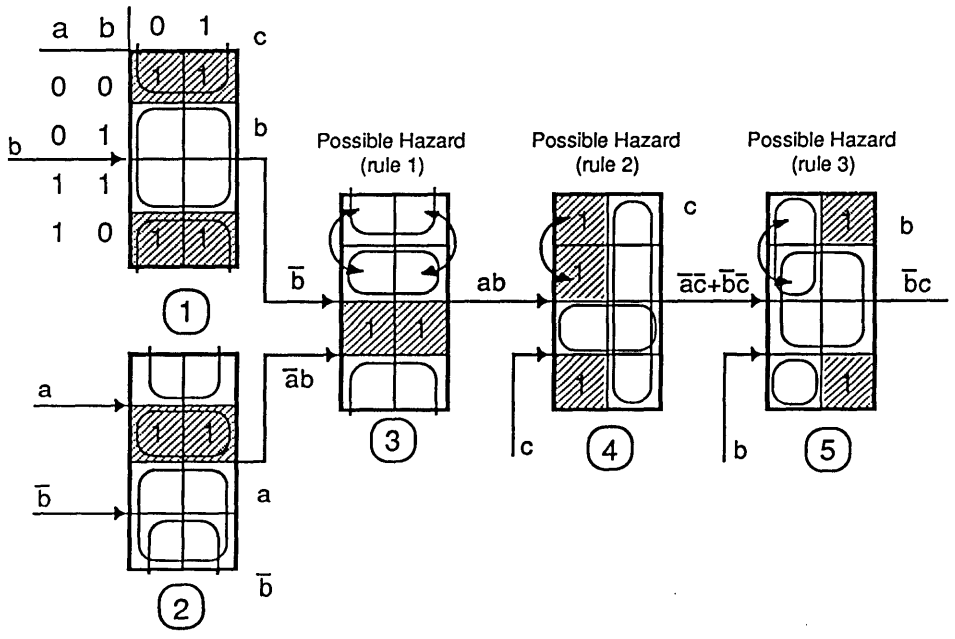


Figure 4-59 (a) Gate-level Karnaugh map for figure 4-58

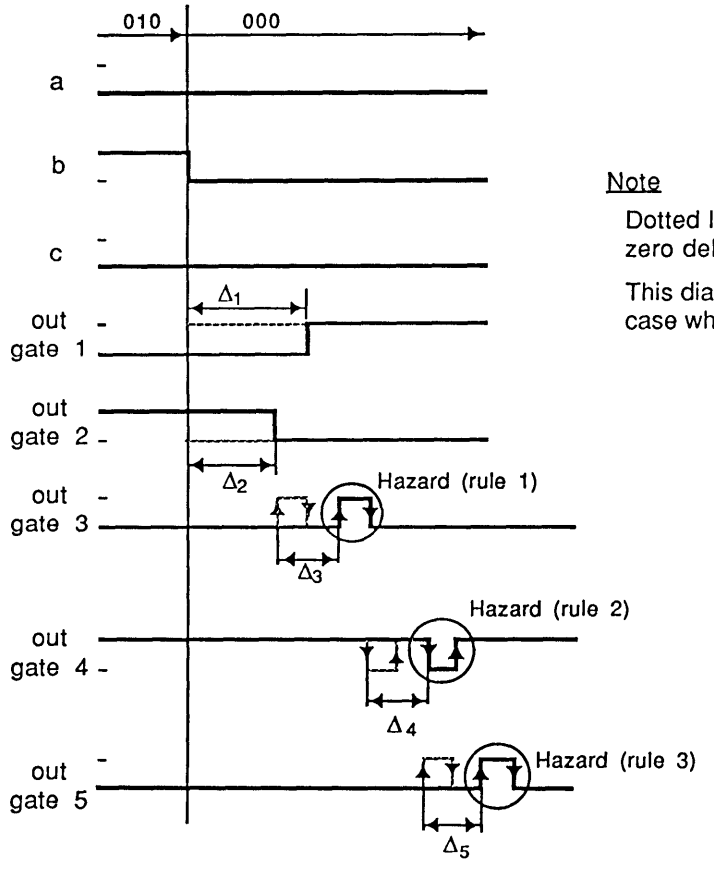


Figure 4-59 (b) Timing diagram showing hazards resulting from a 010 --> 000 transition

Example 4 - Consider the two-level NOR network of Fig. 4-60.

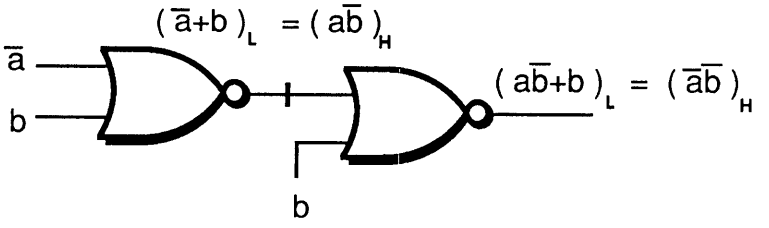


Figure 4-60 Two-level NOR network with a rule (4) hazard

The gate level Karnaugh maps for Fig. 4-60 may be readily drawn as shown in Fig. 4-61(a). Here we see a possible rule (4) hazard in gate 2. The timing diagram illustrates the hazard for a 10 → 11 transition for the case $\Delta_1 > \Delta_2$.

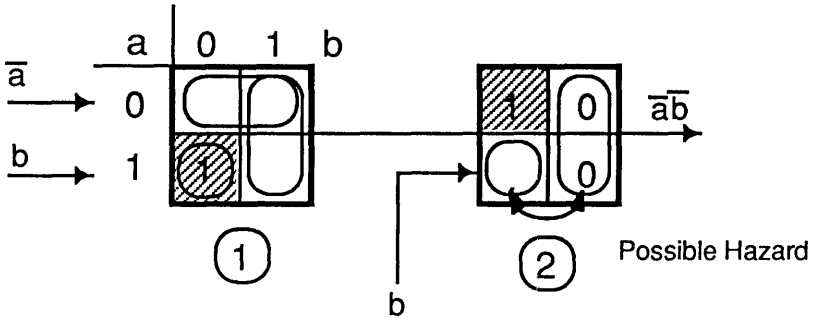
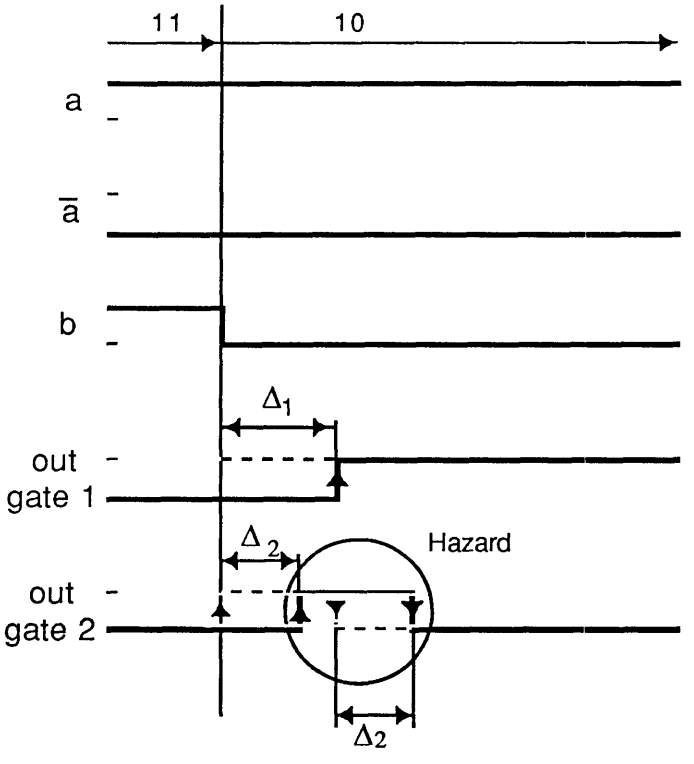


Figure 4-61 (a) Gate-level Karnaugh maps for figure 4-60



Note

Dotted lines represent zero delay gate response

This diagram was drawn for the case where $\Delta_1 > \Delta_2$

Figure 4-61 (b) Timing diagram showing a hazard resulting from a 11 --> 10 transition

Other Map Observations

1. A hazard produced between adjacent inhibiting loops in a source gate is suppressed in the receiving gate if the cells in adjacent inhibiting loops in the receiving gate are surrounded by 0's.

2. When there are adjacent inhibiting loops in a source gate, hazards between 0's become hazards in 1's in the receiving gate.
3. A static hazard can not occur in a gate with only one inhibiting loop. (A loop across the border of a map should not be interpreted as multiple loops.)

Example - Consider the function $f(a,b,c) = \bar{c} + \bar{a}\bar{b} + a\bar{b}$. The NAND network implementing this function and its gate-level (Maxterm) Karnaugh map are shown in Fig. 4-62(a) and (b). Here we recognize a possible hazard in gate #4 for the transition 001 ↔ 101.

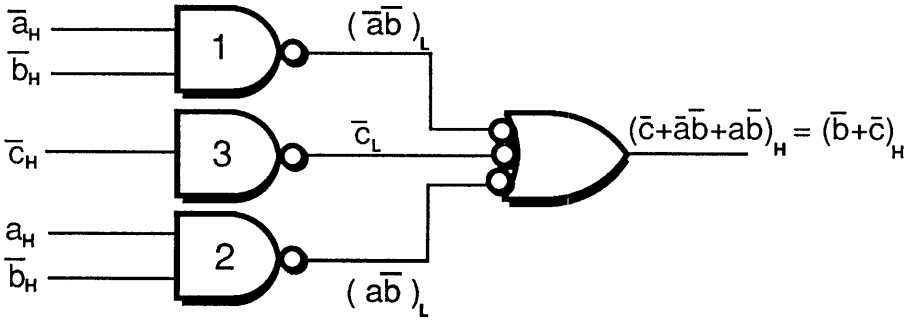


Figure 4-62 (a) NAND network implementing $f(a,b,c)=\bar{c}+\bar{a}\bar{b}+a\bar{b}$

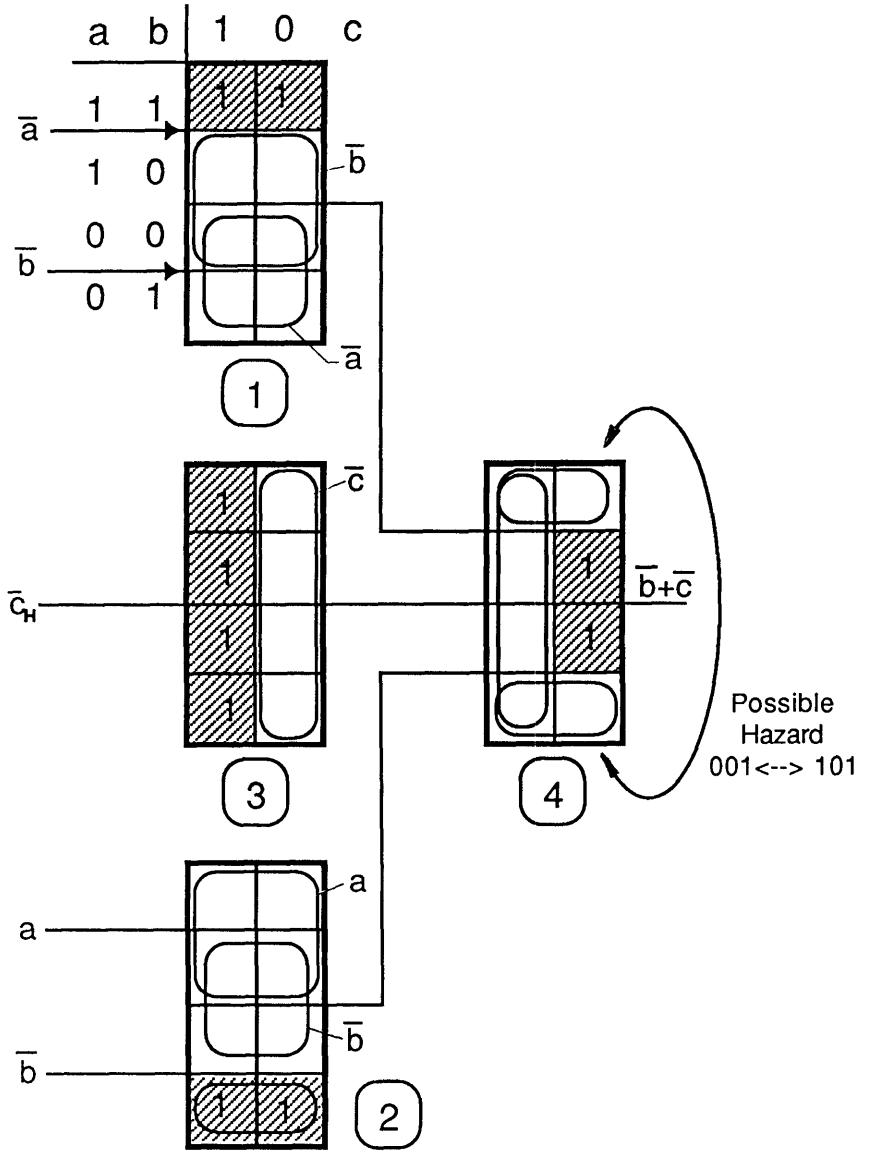
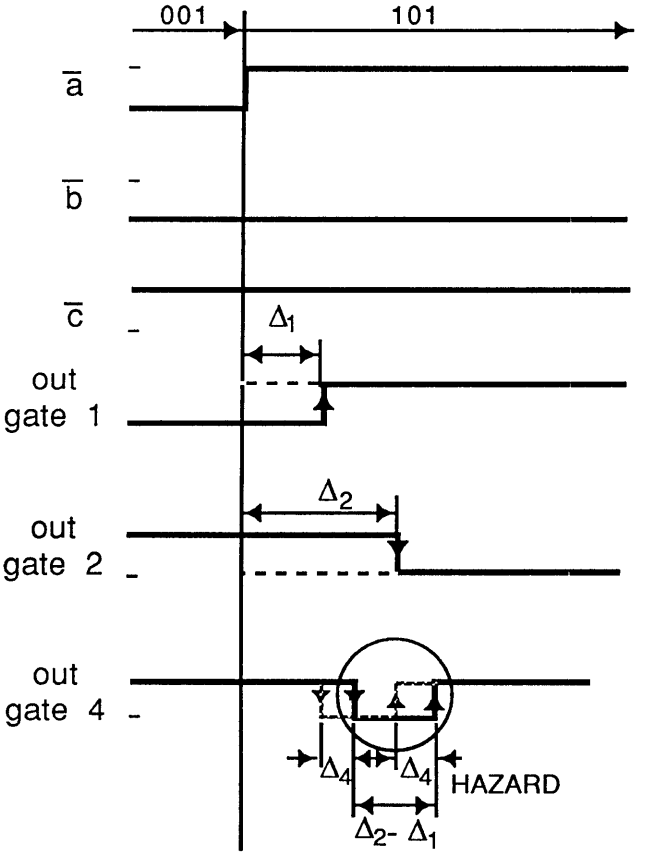


Figure 4-62 (b) Gate-level Karnaugh map

The timing diagram for the transition 001 → 101 is shown in Fig. 4-63. Here Δ_2 is assumed to be greater than Δ_1 ($\Delta_2 > \Delta_1$).



Note

Dotted lines represent zero-delay gate responses

Diagram is drawn for the transition 001 --> 101

Output of gate #3 does not change for the transition pair 001 --> 101, and is high

Figure 4-63 Timing diagram for the NAND network of figure 4-62 (a)

The same functional output may be generated by the NAND network of Fig. 4-64(a) and its gate-level Karnaugh map of Fig. 4-64(b).

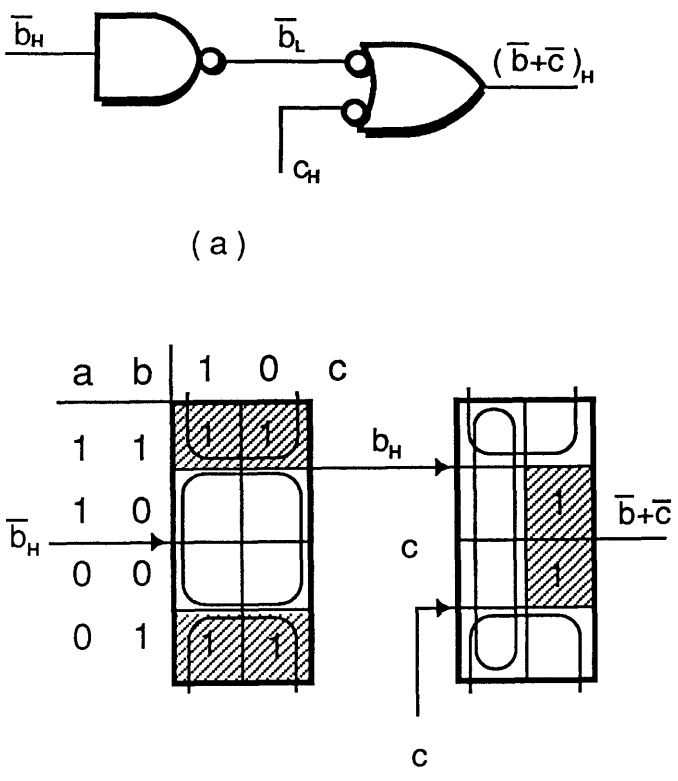


Figure 4-64 Two-level NAND network (a) and its gate-level Karnaugh map (b)

Here there is a single inhibiting loop produced by the upstream gate and consequently a static hazard can not exist. Note - Do not count the implied inhibiting loop produced by the c input.

Dynamic Hazards

If a network's logical equation indicates an output change of 0 → 1 (hereafter called an "α" change) and the output actually changes 0 → 1 → 0 → 1, this is called a dynamic hazard. Similarly, if the network's logical equation indicates an output change 1 → 0 (hereafter called a "β" change) and the output actually changes 1 → 0 → 1 → 0, this

is also called a dynamic hazard. It obviously follows that a static hazard is a prerequisite for a dynamic hazard. Let us examine the conditions that produce these dynamic hazards.

For a NOR gate with inputs a and b as shown, where input b has a static hazard between 0's and a is a simple β change of state, the output will be as shown in Fig. 4-65.

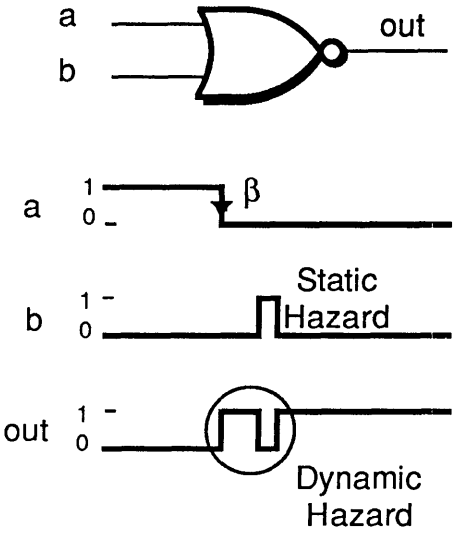


Figure 4-65 NOR gate response to a "β" transition and a static hazard

Thus, we see that a dynamic hazard is created when a static hazard between 0's is NORed with a β transition.

For a NAND gate with inputs a and b as shown in Fig. 4-66, where input b is a static hazard between 1's and a is a simple α change of state, the output will be as shown.

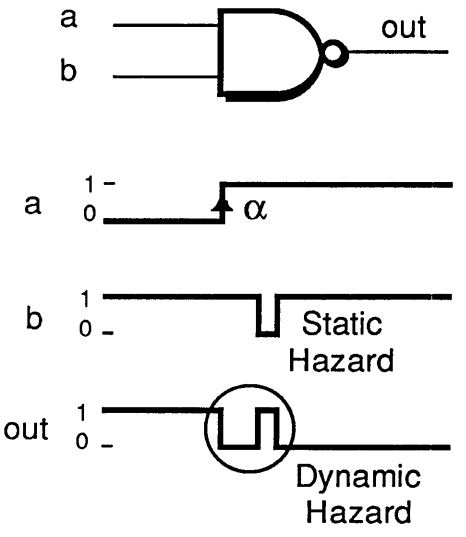


Figure 4-66 NAND gate response to an " α" transition and a static hazard

Thus, a dynamic hazard is created when a static hazard between 1's is NANDed with an α transition.

To illustrate a dynamic hazard in a multiple gate network, consider the circuit of Fig. 4-67. A timing diagram analysis of Fig. 4-67(b) identifies a dynamic hazard in gate 4 and is shown in Fig. 4-68.

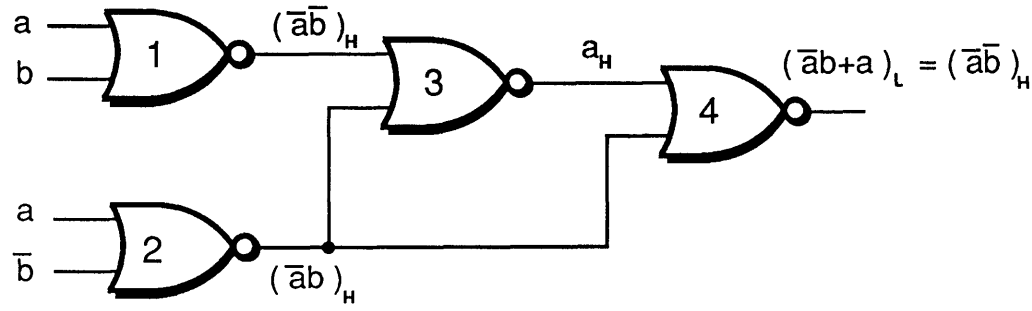


Figure 4-67 (a) Three-level NOR logic containing a dynamic hazard

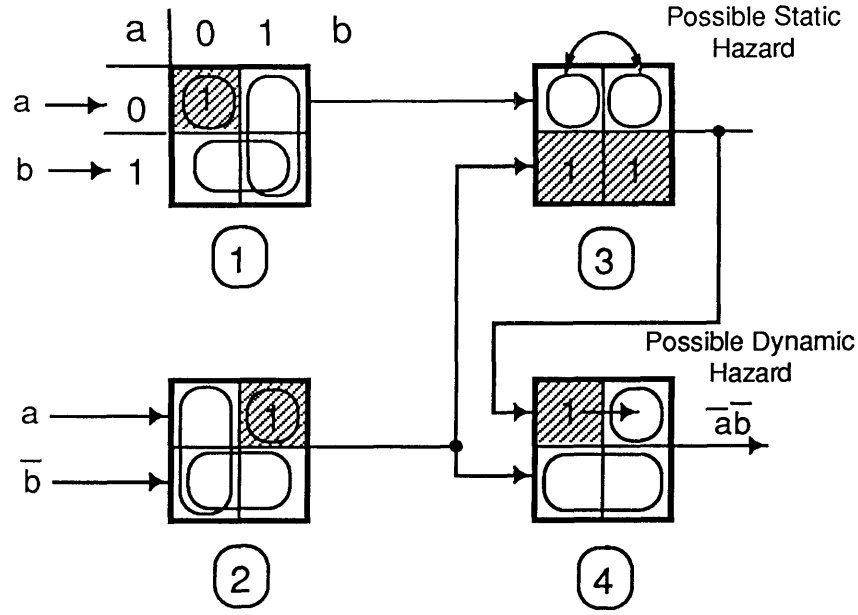
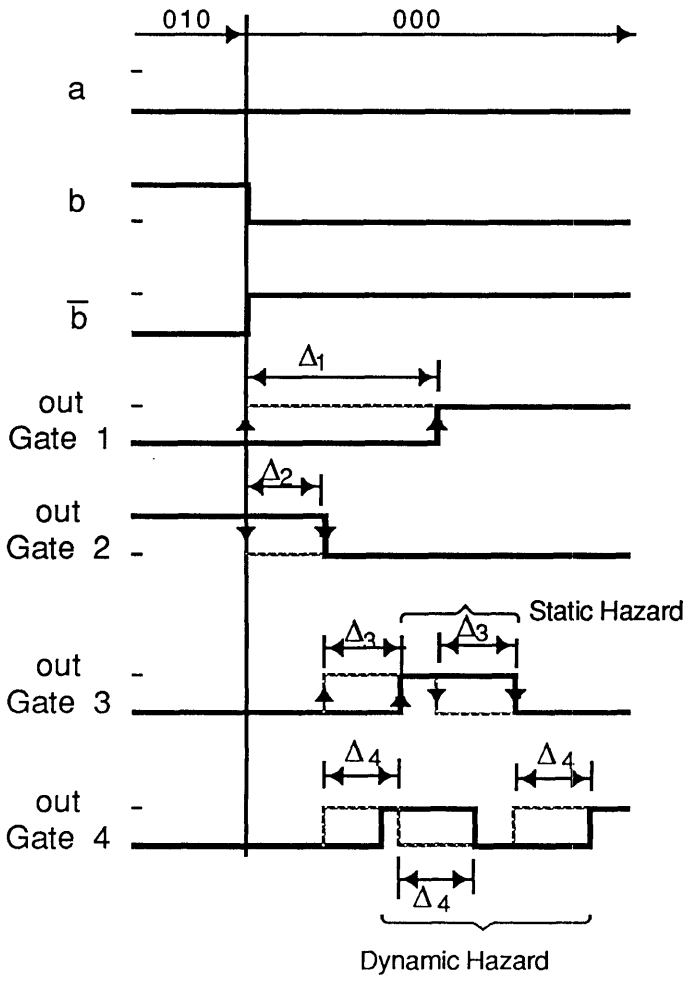


Figure 4-67 (b) Gate-level Karnaugh maps for the logic network

A timing diagram analysis of Fig. 4-67(b) identifies the dynamic hazard in gate 4 and is shown in Fig. 4-68.



Note

Dotted lines represent zero-delay gate response

This diagram was drawn for the case $\Delta_1 > \Delta_2$ & $\Delta_3 > \Delta_4$

Figure 4-68 Timing diagram for the circuit of figure 4-67 (a)

CLOSING REMARKS

This chapter has dealt with a somewhat eclectic collection of subjects supporting the combinational design process under the umbrella title of OTHER COMBINATIONAL LOGIC DESIGN TECHNIQUES. In the previous chapters our attention was centered on the methods for reducing the logical equation's complexity and this certainly is the first and most important step in the design of combinational circuitry. In

this chapter we learned techniques of composite mapping to aid the reduction process by permitting the direct plotting of Boolean functions of mixed form (not minterm exclusively or Maxterm exclusively) without first decomposing reduced terms in the function or transforming these decomposed terms into conjunctive or disjunctive canonical forms. We then expanded our discussion of the Isolation Theorem T-8 to permit the multi-level factoring of Boolean functions with the objective of minimizing the number of gated inputs when implementing a logical function. We then presented a design technique (also called decomposition) which in many cases permits the partitioning of a complex logical function into two component networks, each of significantly reduced complexity, wherein the coupling between the networks is a single logic line. The balance of the chapter was devoted to the analysis of multi-level NOR and NAND logical networks. A technique was presented for transforming a logic diagram into a set of Karnaugh maps wherein each map represented an individual NAND or NOR gate. Interpretative rules were developed to permit the rapid logical interpretation of this gate-level diagram. We concluded the chapter with a discussion of static and dynamic hazards in combinational circuitry caused by the cascaded delay times of the gates themselves. By extending the gate-level mapping used in combinational analysis, we developed interpretive rules to identify potential static and dynamic hazards and showed methods for eliminating some of these problems.

The reader should conclude that combinational logic design is more than simple functional manipulation to achieve a form requiring the fewest number of gates or gated inputs. Other essential considerations deal with the logical and/or physical partitioning of logic networks to facilitate design or physical configurations or the avoidance of static and dynamic hazards in an otherwise logically correct design. The methods presented in this chapter provide sufficient background to permit the designer to complete these tasks. A block diagram illustrating the gross design steps for combinational circuit synthesis is shown in Fig. 4-69.

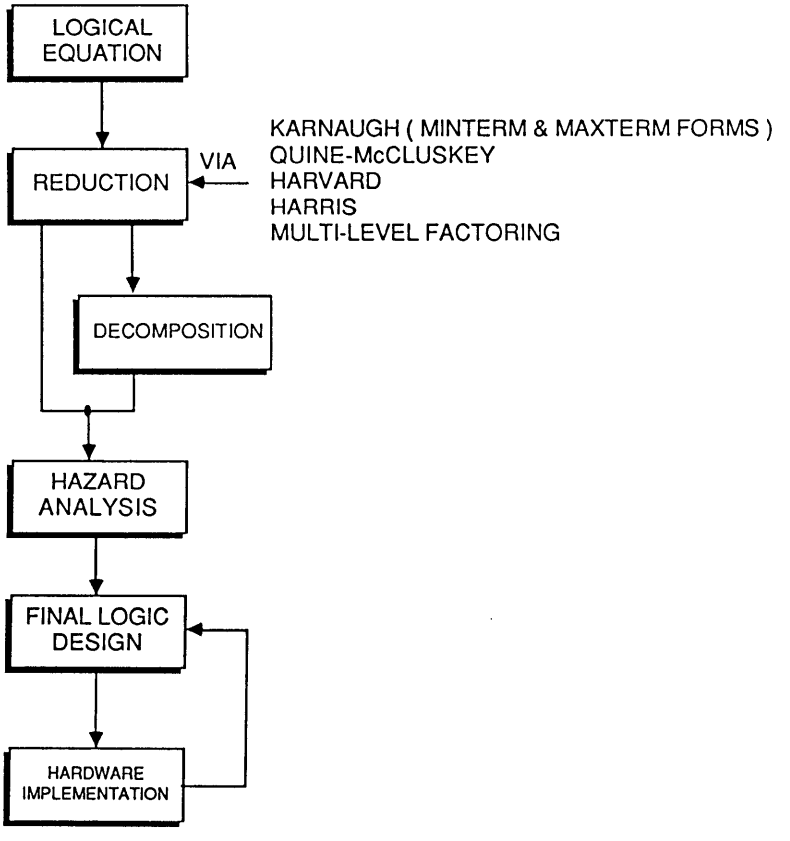


FIGURE 4-69 Gross design steps in the design of combinational logic

REFERENCES

1. Smothers, Carl L.: "A Composite Mapping Technique to Streamline the Use of Karnaugh Maps", Computer Design, pp 128-132, November, 1972.

2. Levine, Robert Irving: "Logical Minimization Beyond the Karnaugh Map", Computer Design, pp 40-43, March, 1967.

3. Maley & Earle: The Logic Design of Transistor Digital Computers, Prentice-Hall Inc. Englewood Cliffs, NJ, 1963.

4. Huffman, D. A.: "The Design and Use of Hazard-Free Switching Networks", Journal of the ACM, vol. 4, pp 47-62, 1957.

5. Erb, T.: "Combinational Hazards in Factored NOR (NAND) Logic", Electronic Progress, vol. X, no. 1 & 2, pp 58-61, 1966.