REPORT T-20 AUGUST, 1975

CSL COORDINATED SCIENCE LABORATORY

ALGEBRAIC NAND-NOR LOGIC DESIGN AN EXPOSITION

FRANZ E. HOHN

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

ALGEBRAIC NAND-NOR LOGIC DESIGN
AN EXPOSITION

by

Franz E. Hohn
Department of Mathematics
University of Illinois at Urbana-Champaign

ACKNOWLEDGMENT

I wish to thank Professors G. Metze and E. S. Davidson for encouraging me to prepare this report from my class notes, and Mr. B. Kumar for a careful reading of the original manuscript.

TABLE OF CONTENTS

1. INTRODUCTION

The two-input NAND function and the two-input NOR function are commonly and conveniently denoted by such symbols as the stroke:

$$a|b = \overline{ab} = \bar{a} + \bar{b} \quad \text{(NAND)}$$

and the arrow:

$$a{\downarrow}b = \overline{a+b} = \bar{a}\,\bar{b} \quad \text{(NOR)}.$$

It is then routine to transform a given Boolean expression into a form employing only two-input NANDs or NORs or a mixture of these by using, in addition to the definitions, the relations

$$a = \bar{a}|\bar{a}, \quad a+b = \bar{a}|\bar{b},$$

and

$$a = \bar{a}{\downarrow}\bar{a}, \quad ab = \bar{a}{\downarrow}\bar{b}.$$

To illustrate, one way of transforming the exclusive-OR is this:

$$\begin{aligned} a \oplus b = a\bar{b} + \bar{a}b &= \overline{a\bar{b}}\,|\,\overline{\bar{a}b} \\ &= (a|\bar{b})|(\bar{a}|b) \\ &= (a|(b|b))|((a|a)|b). \end{aligned}$$

This requires five NAND-gates, but four will suffice. We have

$$\begin{aligned} a|(b|b) &= \bar{a} + b \\ &= \bar{a} + ab \\ &= a|\overline{ab} \\ &= a|(a|b). \end{aligned}$$

If we interchange a and b in this formula and use the fact that the stroke operation is commutative, we obtain the equivalent identity

$$(a|a)|b = (a|b)|b.$$

Substituting from these two identities into the previous expression for $a \oplus b$, we obtain

$$a \oplus b = (a|(a|b))|((a|b)|b),$$

a form which requires only four NAND-gates because of the repetition of the function $(a|b)$.

Dually, the both-or-neither (equivalence) function $ab + \overline{a}\,\overline{b} = a \odot b$ is given by

$$a \odot b = (a \downarrow (a \downarrow b)) \downarrow ((a \downarrow b) \downarrow b).$$

The circuits are shown in Figure 1.1.

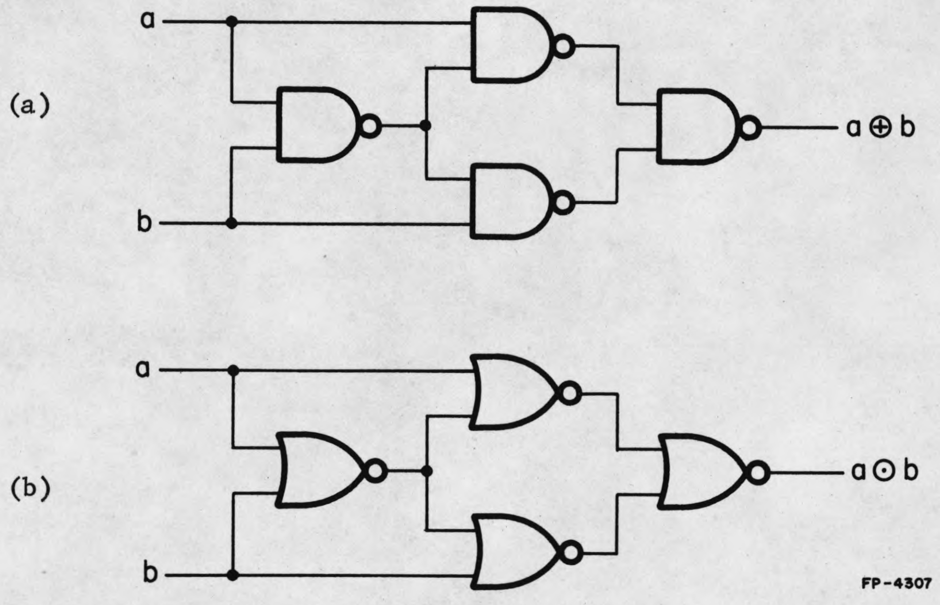


Figure 1.1. (a) EXCLUSIVE-OR circuit
(b) EQUIVALENCE circuit

This particular treatment of a familiar example suggests that identities involving NAND and NOR functions can be useful for the direct transformation of switching functions expressed in terms of these functions without appeal to their AND-OR-NOT equivalents. This is indeed the case, but the development of such identities requires a suitable notation for multiple-input NAND and NOR functions. The stroke type of symbolism does not generalize properly and repeated overbars or primes to denote negation become awkward to read and to manipulate. Since we will be concerned with functions of arbitrarily many arguments, a natural procedure is to use functional notation for the development of the algebra of these functions.

Throughout this development of multiple-input NAND and NOR algebra, we shall appeal freely to the definitions and to corresponding relations expressed in terms of AND, OR, and NOT. However, it is important to remember that a major objective in developing such an algebra is to make possible avoidance of unnecessary transformations of this kind.

An alternative, postulational development of NAND-NOR algebra that does not appeal at all to the operations AND, OR, and NOT has also been developed. Such an approach is both interesting and mathematically instructive, but from a practical point of view, it is more efficient to allow appropriate interplay between the two systems.

Another approach to NAND-NOR logic design is the map method given by G. Maley and J. Earle in <u>The Logic Design of Transistor Digital Computers</u>, Englewood Cliffs, N. J., Prentice-Hall, 1963. A number of examples in the present exposition use functions employed by Maley and Earle in their examples. These are appropriately referenced so the reader can compare the two procedures if he wishes.

2. NOTATION AND DEFINITIONS

As we have noted, there is no standard functional notation for the generalized NAND and NOR functions. In what follows, we shall use M to denote the NAND function and m to denote the NOR function. These designations draw attention to the maxterm character of NAND and the minterm character of NOR, respectively. The basic definitions are these: For all switching functions $f_1, f_2, \ldots, f_k$ of n variables, and for all possible integers k,

(2.1a)
$$M(f_1) = \overline{f}_1$$
$$M(f_1, f_2, \ldots, f_k) = \overline{f_1 f_2 \cdots f_k} = \overline{f}_1 + \overline{f}_2 + \cdots + \overline{f}_k$$

and

(2.1b)
$$m(f_1) = \overline{f}_1,$$
$$m(f_1, f_2, \ldots, f_k) = \overline{f_1 + f_2 + \cdots + f_k} = \overline{f}_1 \overline{f}_2 \cdots \overline{f}_k \ .$$

Note that M(p), where p is a minterm, is a maxterm and that m(s), where s is a maxterm, is a minterm.

Since the terms of a sum and the factors of a product may be permuted arbitrarily, it follows from these definitions that <u>each of the operations NAND and NOR is commutative</u> in the sense that its arguments may be permuted arbitrarily.

From the definitions we have at once the <u>OR-to-NAND and AND-to-NOR transformation laws</u>:

(2.2a)
$$f_1 + f_2 + \cdots + f_k = M(\overline{f}_1, \overline{f}_2, \ldots, \overline{f}_k)$$

and

(2.2b)
$$f_1 f_2 \cdots f_k = m(\overline{f}_1, \overline{f}_2, \ldots, \overline{f}_k).$$

In words, a sum is the NAND of the complements of its terms and a product is the NOR of the complements of its factors.

The formulas (2.2) have important special cases. If

$$X = (x_1, x_2, \ldots, x_n)$$

and if

$$f(X) = f_1 + f_2 + \cdots + f_k,$$

where the f_j are products of switching functions of $x_1, x_2, \ldots, x_n$ (often products of literals):

$$f_i = g_{i1} g_{i2} \cdots g_{ir_i},$$

we have, by (2.1a),

$$\overline{f}_i = M(g_{i1}, g_{i2}, \ldots, g_{ir_i})$$

so (2.2a) becomes the familiar sum-of-products to NAND transformation:

(2.3a) $$\begin{aligned} &g_{11} g_{12} \cdots g_{1r_1} + g_{21} g_{22} \cdots g_{2r_2} + \cdots + g_{k1} g_{k2} \cdots g_{kr_k} \\ &= M[M(g_{11}, g_{12}, \ldots, g_{1r_1}), M(g_{21}, g_{22}, \ldots, g_{2r_2}), \ldots, \\ &\qquad M(g_{k1}, g_{k2}, \ldots, g_{kr_k})]. \end{aligned}$$

For example, using these observations, we can write at once,

$$x_1 x_2 + x_2 x_3 + x_3 x_1 = M[M(x_1, x_2), M(x_2, x_3), M(x_3, x_1)].$$

Similarly, if

$$f(X) = f_1 f_2 \ldots f_k,$$

where the f_i are sums of switching functions of $x_1, x_2, \ldots, x_n$ (often sums of literals):

$$f_i = g_{i1} + g_{i2} + \cdots + g_{ir_i},$$

then by (2.1b) we have

$$\overline{f}_i = m(g_{i1}, g_{i2}, \ldots, g_{ir_i}),$$

so (2.2b) becomes the familiar product-of-sums to NOR transformation:

(2.3b) $$(g_{11} + g_{12} + \cdots + g_{1r_1})(g_{21} + g_{22} + \cdots + g_{2r_2}) \cdots (g_{k1} + g_{k2} + \cdots + g_{kr_k})$$
$$= m[m(g_{11}, g_{12}, \ldots, g_{1r_1}), m(g_{21}, g_{22}, \ldots, g_{2r_2}), \ldots,$$
$$m(g_{k1}, g_{k2}, \ldots, g_{kr_k})].$$

For example, by (2.3b) we have at once

$$x_1(x_2 + x_3)(\overline{x}_2 + \overline{x}_3) = m[m(x_1), m(x_2, x_3), m(\overline{x}_2, \overline{x}_3)].$$

It is useful to define, in analogy to the Σ and π notations for sums and products,

$$\underset{i=1}{\overset{k}{M}}(f_i) = M(f_1, f_2, \ldots, f_k)$$

and

$$\underset{i=1}{\overset{k}{m}}(f_i) = m(f_1, f_2, \ldots, f_k).$$

With the aid of this notation, the cumbersome identities (2.3) may be rewritten thus:

(2.3a)' $$\sum_{i=1}^{k} \left(\prod_{j=1}^{r_i} g_{ij} \right) = \underset{i=1}{\overset{k}{M}} \left(\underset{j=1}{\overset{r_i}{M}} (g_{ij}) \right)$$

and

(2.3b)' $$\prod_{i=1}^{k} \left(\sum_{j=1}^{r_i} g_{ij} \right) = \underset{i=1}{\overset{k}{m}} \left(\underset{j=1}{\overset{r_i}{m}} (g_{ij}) \right).$$

From the definitions we have at once the following four identities which hold for all switching functions $f_1, f_2, \ldots, f_k$ and for all positive integers k:

(2.4a) $$M(0,f_1,f_2,\ldots,f_k) = 1,$$

(2.4b) $$m(1,f_1,f_2,\ldots,f_k) = 0,$$

and

(2.5a) $$M(1,f_1,f_2,\ldots,f_k) = M(f_1,f_2,\ldots,f_k),$$

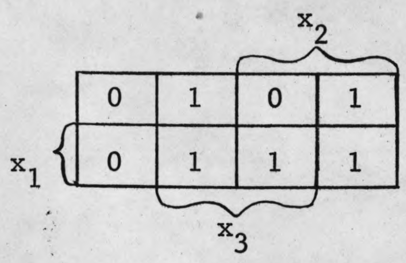
(2.5b) $$m(0,f_1,f_2,\ldots,f_k) = m(f_1,f_2,\ldots,f_k).$$

These identities are frequently useful in the derivation of others.

Throughout this paper, NAND-NOR formulas appear in pairs differing only in that M and m, and 0 and 1, are interchanged throughout. Each formula of such a pair will be called the dual of the other. If AND and OR are also used, they too must be interchanged throughout.

3. NAND AND NOR EXPRESSIONS FROM MAPS

Consider the function defined by the map



from which we have the minimal forms

$$f(X) = \bar{x}_2x_3 + x_2\bar{x}_3 + \left\{\begin{matrix} x_1x_2 \\ x_1x_3 \end{matrix}\right\}$$

and

$$f(X) = (x_2 + x_3)(x_1 + \bar{x}_2 + \bar{x}_3).$$

These can be transformed into M and m forms respectively by (2.3), the g_{ij} now being literals. On the other hand, since exactly the same literals appear in the M and m forms as appear in the minimal sum-of-products and product-of-sums forms respectively, the M and m expansions can be, and should be, written directly from the map. In the case of the present example, by inspection of the map we would write at once

$$f(X) = M\left[M(\bar{x}_2,x_3),M(x_2,\bar{x}_3), \left\{\begin{matrix} M(x_1,x_2) \\ M(x_1,x_3) \end{matrix}\right\}\right]$$

and

$$f(X) = m[m(x_2,x_3),m(x_1,\bar{x}_2,\bar{x}_3)].$$

The results obtained in this way from maps of functions (or by transformation of the results of any of the several minimization procedures) may then be further transformed by appropriate NAND-NOR identities until the most useful end results are obtained. This will be illustrated in following sections.

The NAND and NOR representations of f obtained in this way are necessarily minimal two-stage representations of f. Indeed, if there were a shorter two-stage representation, it would lead immediately to a shorter sum-of-products or product-of-sums representation of f.

As another example, consider the function defined by the map of Figure 3.1. Here the minimal two-stage NAND and NOR representations are

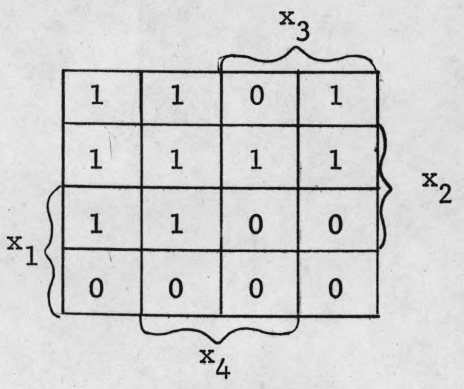


Figure 3.1

$$f(X) = M[M(\overline{x}_1,x_2),M(\overline{x}_1,\overline{x}_3),M(\overline{x}_1,\overline{x}_4),M(x_2,\overline{x}_3)]$$

and

$$f(X) = m[m(\overline{x}_1,x_2),m(\overline{x}_1,\overline{x}_3),m(x_2,\overline{x}_3,\overline{x}_4)].$$

The corresponding minimal disjunctive and conjunctive forms may be factored, among other ways, as follows:

$$f(X) = \overline{x}_1(x_2+\overline{x}_3+\overline{x}_4) + x_2\overline{x}_3$$

and

$$f(X) = (\overline{x}_1+x_2\overline{x}_3)(x_2+\overline{x}_3+\overline{x}_4).$$

Applying the appropriate transformation rules, we have then

$$f(X) = M[M\{\bar{x}_1, M(\bar{x}_2, x_3, x_4)\}, M(x_2, \bar{x}_3)]$$

and

$$f(X) = m[m\{\bar{x}_1, m(\bar{x}_2, x_3)\}, m(x_2, \bar{x}_3, \bar{x}_4)].$$

One would expect to be able to effect a transformation of the initially written M and m forms to these factored forms without first reverting to AND-OR-NOT expressions. A set of factoring rules that will accomplish this is given in the next section.

4. THE BASIC FACTORING RULES

The rules given in this section will be derived by first transforming M and m expressions to AND-OR-NOT equivalents, rearranging these, and then transforming back to NAND and NOR expressions by the rules of Section 2.

A frequently encountered type of M-expression in which factoring is possible is the following:

$$f(X) = M[M(f,g_1),M(f,g_2),\ldots,M(f,g_k),h_1,h_2,\ldots,h_q].$$

We have at once, by the rules of Section 2,

$$f(X) = fg_1 + fg_2 + \cdots + fg_k + \bar{h}_1 + \bar{h}_2 + \cdots + \bar{h}_q$$

so

$$f(X) = f(g_1 + g_2 + \cdots + g_k) + \bar{h}_1 + \bar{h}_2 + \cdots + \bar{h}_q$$
$$= M[M\{f,M(\bar{g}_1,\bar{g}_2,\ldots,\bar{g}_k)\},h_1,h_2,\ldots,h_q].$$

Let us use h^* and h^+ to denote respectively "any non-negative integral number of arbitrary arguments" and "any positive integral number of arguments" h_i. That is, h^* may be an empty set of arguments, but h^+ may not. Moreover, whenever h^* or h^+ is repeated in a given identity, it stands for the same set of arguments at each appearance. Then the preceding computations imply the identity

(4.1a) $$M[M(f,g_1),M(f,g_2),\ldots,M(f,g_k),h^*]$$
$$= M[M\{f,M(\bar{g}_1,\bar{g}_2,\ldots,\bar{g}_k)\},h^*].$$

In exactly dual fashion, we obtain

(4.1b) $$m[m(f,g_1),m(f,g_2),\ldots,m(f,g_k),h^*] = m[m\{f,m(\overline{g}_1,\overline{g}_2,\ldots,\overline{g}_k)\},h^*].$$

The computations assume that h^* is non-empty. The reader should show that (4.1a) and (4.1b) hold even if h^* is empty, that is, even if no arguments h are present.

Note that the reductions effected in the second example of the preceding section could have been effected by applying these two factoring rules.

The two preceding rules generalize to the following, which may be proved in precisely the same way:

(4.2a) $$\begin{aligned}&M[M(f_1,f_2,\ldots,f_r,g_{11},g_{12},\ldots,g_{1r_1}),\\ &\quad M(f_1,f_2,\ldots,f_r,g_{21},g_{22},\ldots,g_{2r_2}),\ldots,\\ &\qquad M(f_1,f_2,\ldots,f_r,g_{k1},g_{k2},\ldots,g_{kr_k}),h^*]\\ &= M[M(f_1,f_2,\ldots,f_r,M\{M(g_{11},g_{12},\ldots,g_{1r_1}),M(g_{21},g_{22},\ldots,g_{2r_2}),\\ &\qquad \ldots,M(g_{k1},g_{k2},\ldots,g_{kr_k})\}),h^*].\end{aligned}$$

The corresponding m-relation (4.2b) is obtained by replacing M by m throughout in (4.2a).

Letting g_i^+ denote an arbitrary positive number of arguments $g_{i1},g_{i2},\ldots,g_{ir_i}$ and using h^* as before, the cumbersome identity (4.2a) can be rewritten thus:

(4.2a)' $$M[M(f^+,g_1^+),M(f^+,g_2^+),\ldots,M(f^+,g_k^+),h^*] = M[M(f^+,M\{M(g_1^+),M(g_2^+),\ldots,M(g_k^+)\}),h^*],$$

and similarly for its dual.

It is important to note how the factorings (4.2) affect fan-in and fan-out: The fan-out of each f_i is reduced from k to 1 while fan-in for the final NAND-gate (written first in the formula) is reduced by k-1.

As a further illustration of these rules for factoring, we design a full adder using only 2-input NAND-gates. The adder is defined by the table and maps of Figure 4.1.

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

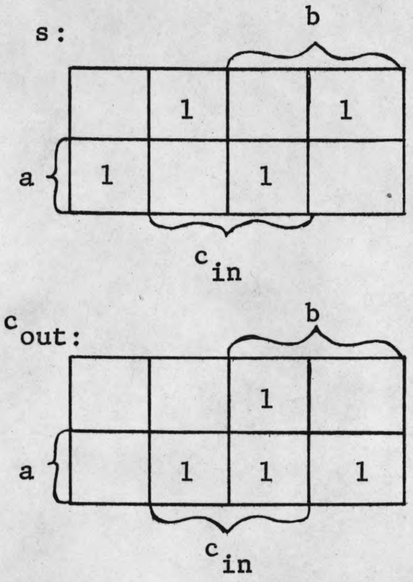


Figure 4.1. Definition of Full Adder

From the maps, we have at once the minimal two-stage expressions for s and c_{out}:

$$s = M[M(\overline{a},\overline{b},c_{in}),M(\overline{a},b,\overline{c}_{in}),M(a,\overline{b},\overline{c}_{in}),M(a,b,c_{in})]$$

and

$$c_{out} = M[M(a,b),M(a,c_{in}),M(b,c_{in})].$$

By the commutativity of M and the factoring rule (4.2a), we have

$$s = M[M(c_{in},M\{M(\bar{a},\bar{b}),M(a,b)\}),M(\bar{c}_{in},M\{(\bar{a},b),M(a,\bar{b})\})]$$

and

$$c_{out} = M[M\{c_{in},M(\bar{a},\bar{b})\},M(a,b)].$$

These factorings were chosen to permit sharing of the outputs $M(\bar{a},\bar{b})$ and $M(a,b)$.

Assuming that complements of inputs are available, the corresponding circuit diagram is as shown in Figure 4.2. Note that only two-input gates are employed. Other implementations of the full adder will appear in later sections.

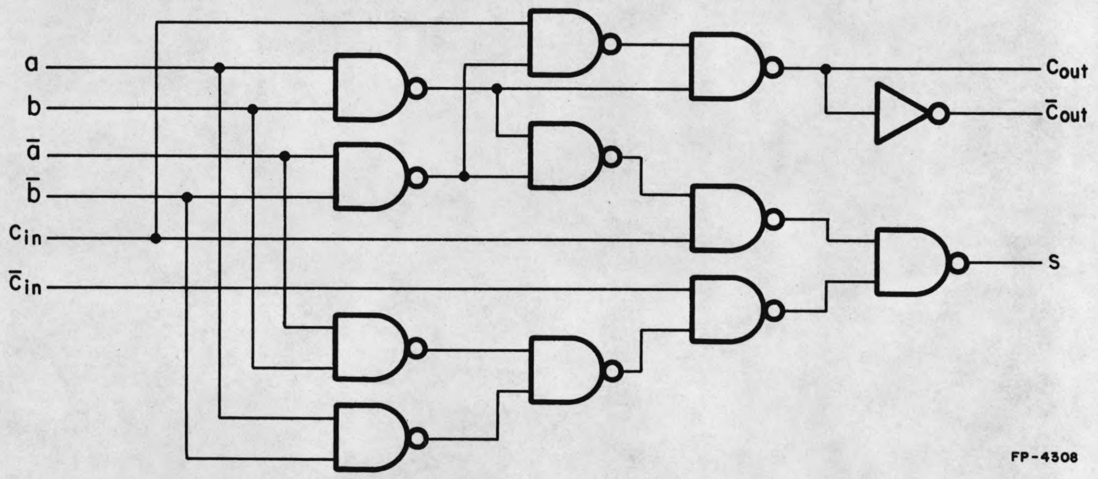


Figure 4.2. NAND Full Adder (23 inputs, 12 gates, 4 stages)

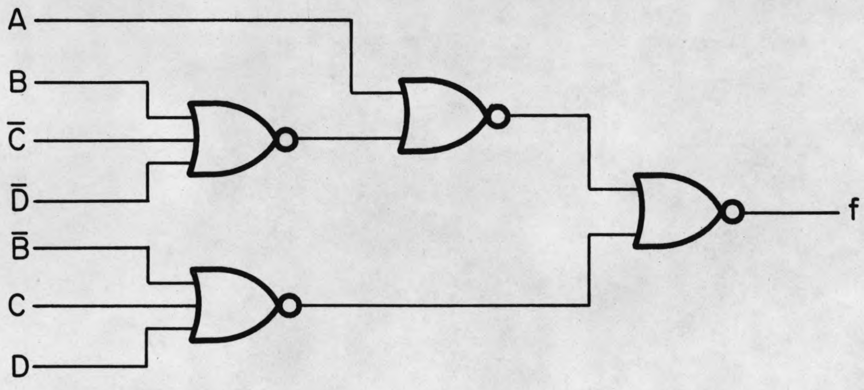
As a second example, consider the following function, a variation of an example of Maley and Earle (p. 127):

$$f(A,B,C,D) = (A+\overline{B})(A+C)(A+D)(\overline{B}+C+D).$$

Here we have immediately, by (4.1b),

$$f(A,B,C,D) = m[m\{A,m(B,\overline{C},\overline{D})\},m(\overline{B},C,D)].$$

The corresponding circuit appears in Figure 4.3.



FP-4309

Figure 4.3. A NOR Circuit

Expanding the product-of-sums form of f judiciously, we have

$$f(A,B,C,D) = A(\overline{B}+C+D) + \overline{B}CD$$

so, alternatively,

$$f(A,B,C,D) = M[M\{A,M(B,\overline{C},\overline{D})\},M(\overline{B},C,D)].$$

An interesting fact is that the M-expression for f could have been obtained from the m-expression for f simply by replacing m by M throughout.

The explanation for this is the fact that the equation

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

is self dual.

In general, let $f(X;M)$ denote a purely NAND function of the x_i and their complements and let $f(X;m)$ denote the same expression with M replaced by m throughout. Also let $\tilde{f}(X;\mathrm{AND},\mathrm{OR},\mathrm{NOT})$ denote the AND-OR-NOT translation of $f(X,M)$ and let $\tilde{f}(X;\mathrm{OR},\mathrm{AND},\mathrm{NOT})$ denote the same expression with AND and OR interchanged throughout. Then because of the rules for transforming NAND and NOR expressions into AND-OR-NOT expressions and vice versa,

$$f(X;M) = f(X;m) \text{ iff } \tilde{f}(X;\mathrm{AND},\mathrm{OR},\mathrm{NOT}) = \tilde{f}(X;\mathrm{OR},\mathrm{AND},\mathrm{NOT}).$$

For example, because

$$\overline{\overline{ab}\cdot\overline{bc}\cdot\overline{ca}} = \overline{\overline{a+b}+\overline{b+c}+\overline{(c+a)}},$$

we have

$$M(M(a,b),M(b,c),M(c,a)) = m(m(a,b),m(b,c),m(c,a)).$$

5. IMPLEMENTATION WITH MIXED LOGIC

Often a mixture of NAND, NOR, AND, and OR elements is algebraically natural and in certain cases leads to representations that permit the use of specialized types of integrated circuits. We give some examples.

Example 5.1: Consider the function

$$f(A,B,C) = (A+\overline{B})(A+C)(A+D)(\overline{B}+\overline{C}+D) = (A+\overline{B}CD)(\overline{BC}+D),$$

the second form of which yields

$$\begin{aligned} f(A,B,C) &= m[m(A,\overline{B}CD),m(\overline{BC},D)] \\ &= m[m\{A,m(B,\overline{CD})\},m(\overline{BC},D)] \\ &= m[m\{A,m(B,M(C,D))\},m\{M(B,C),D\}]. \end{aligned}$$

The circuit diagram is shown in Figure 5.1.

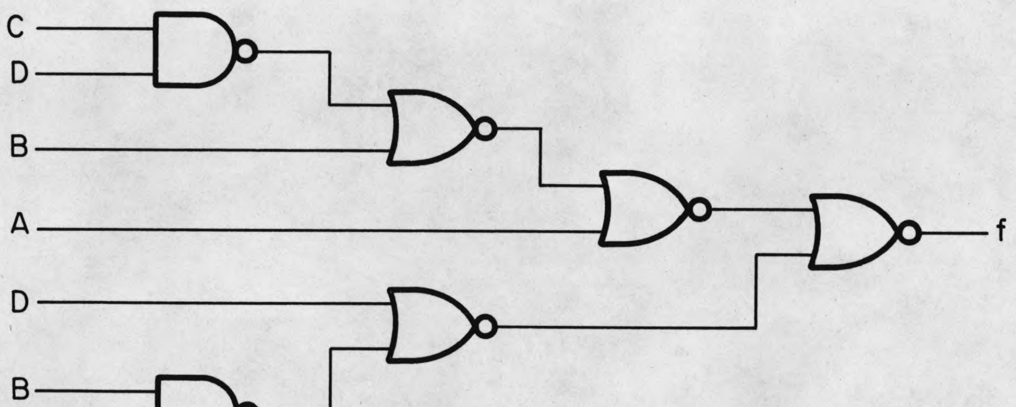


Figure 5.1. A NAND-NOR Circuit.

The point of the example is that by mixing NANDs and NORs we readily obtain an implementation of f that requires no complemented inputs. Moreover, this implementation is more economical than is a purely NAND implementation (Maley and Earle, p. 127).

Example 5.2: If complements of inputs are not available, the implementation of

$$f(X) = (\bar{x}_1 + \bar{x}_2 + \bar{x}_3)(x_4 + \bar{x}_5 + \bar{x}_6)$$

may be accomplished as a mixed logic circuit, thus:

$$f(X) = m[x_1 x_2 x_3, m\{x_4, M(x_5, x_6)\}].$$

See Figure 5.2, in which AND is implemented as NAND-NOT.

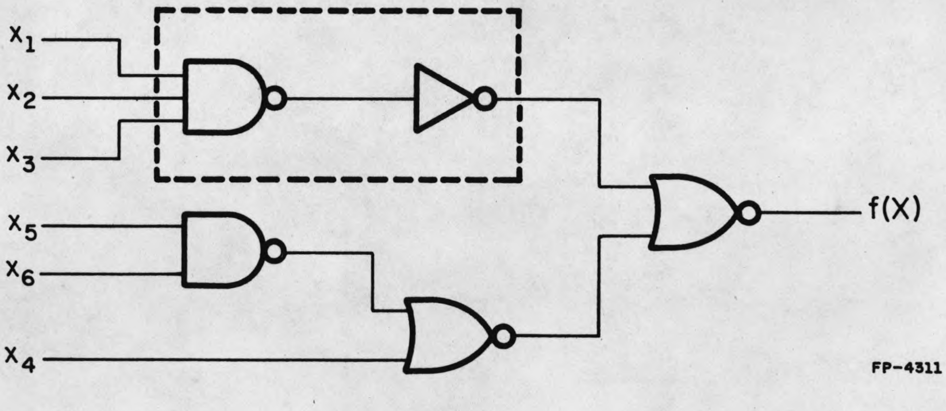


Figure 5.2. A NAND-NOR-AND Circuit.

Example 5.3: Consider the function

$$f(A,B,C) = (\bar{A} + B + C)(A + BC + \bar{B}\bar{C}).$$

If complements are available, we write at once

$$f(A,B,C) = m[m(\overline{A},B,C),m\{A,m(\overline{B},\overline{C}),m(B,C)\}].$$

The corresponding circuit requires five NOR-gates with a total of twelve inputs.

If complements are not available, a NOR-AND implementation is appropriate:

$$\begin{aligned} f(A,B,C) &= (\overline{A}+(B+C))(A+BC+\overline{B}\,\overline{C}) \\ &= m[A\cdot\overline{B+C},m\{A,BC,m(B,C)\}] \\ &= m[A\cdot m(B,C),m\{A,BC,m(B,C)\}]. \end{aligned}$$

The circuit is given in Figure 5.3.

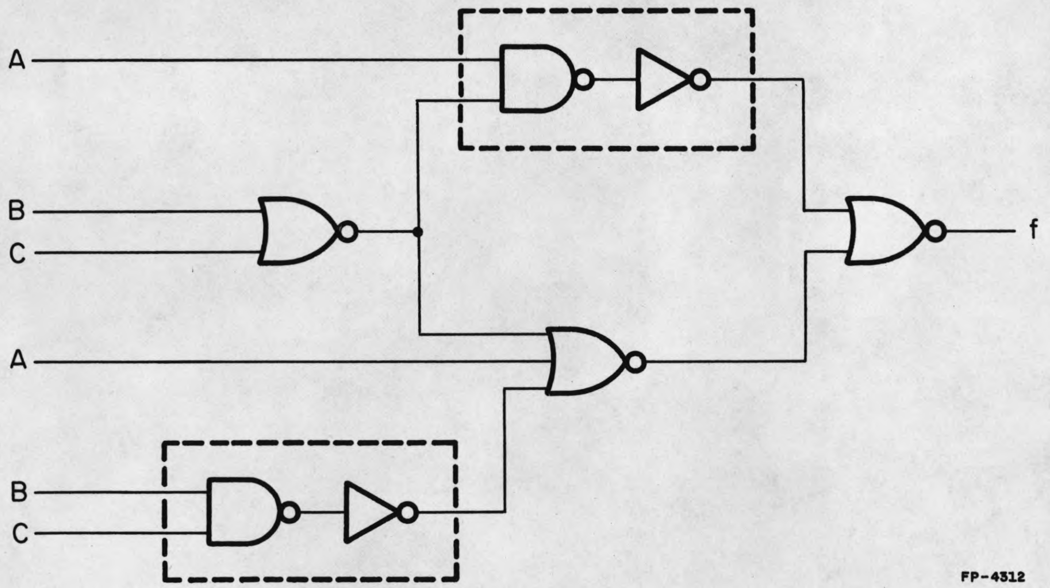


Figure 5.3. A NOR-AND Circuit.

If we rewrite f(A,B,C) in the form

$$f(A,B,C) = \overline{A}\,\overline{B}\,\overline{C} + A(B+C) + BC,$$

so

$$f(A,B,C) = M[A + (B + C), M(A, B + C), M(B,C)],$$

we obtain the NAND-OR implementation of Figure 5.4.

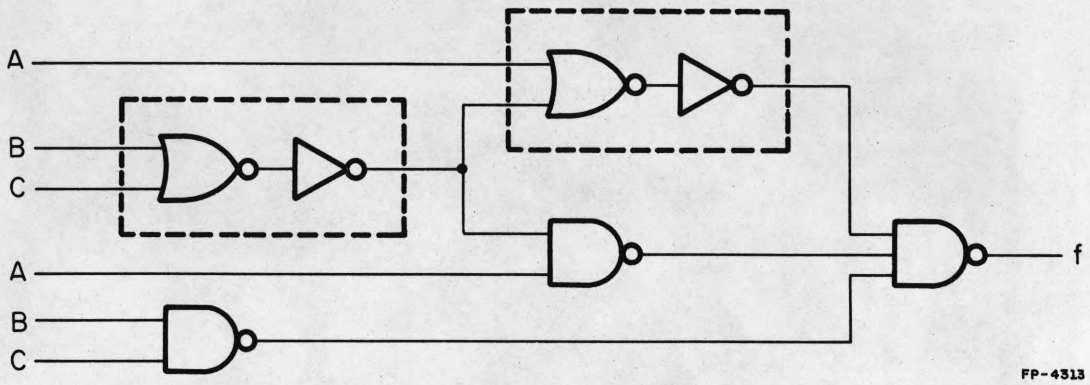


Figure 5.4. A NAND-OR Circuit.

Other implementations of this function appear in Maley and Earle, page 122. The several treatments of this one function illustrate the great flexibility inherent in NAND-NOR algebra.

Example. 5.4: Consider again the sum function of a full adder:

$$s = \overline{a}\,\overline{b}\,c + \overline{a}b\overline{c} + a\,\overline{b}\,\overline{c} + abc.$$

Assume complements not available. We have

$$s = \overline{a+b} \cdot c + \overline{a+c} \cdot b + \overline{b+c} \cdot a + abc$$

so

$$s = M[M\{m(a,b),c\},M\{m(a,c),b\},M\{m(b,c),a\},M(a,b,c)].$$

The circuit is shown in Figure 5.5.

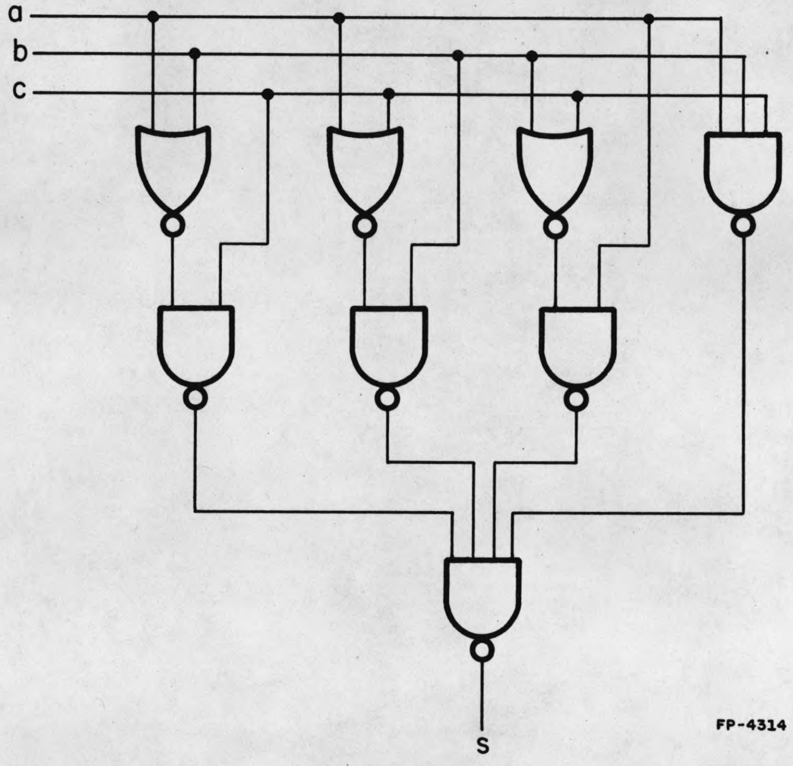


Figure 5.5. Sum Digit for Full Adder (19 inputs, 8 gates, 3 stages).

The same function s was generated in Section 4 with 18 inputs, 9 two-input NAND-gates and 4 stages, assuming complements to be available.

The two-stage implementation

$$s = M[M(\bar{a},\bar{b},c),M(\bar{a},b,\bar{c}),M(a,\bar{b},\bar{c}),M(a,b,c)]$$

requires only 16 inputs and 5 multiple-input gates, assuming complements to be available.

Another implementation of s will be given in Section 6.

6. THE LAWS OF REDUNDANCY

Of particular value in the transformation of NAND and NOR expressions for design purposes are the readily derived laws of redundancy:

(6.1a) $M(f,\overline{g},h^*) = M[f,M(f,g),h^*]$,

(6.2a) $M[f,M(f,g_1,g_2,\ldots,g_k),h^*] = M[f,M(g_1,g_2,\ldots,g_k),h^*]$,

(6.3a) $M[f,m(\overline{f},g_1,g_2,\ldots,g_k),h^*] = M[f,m(g_1,g_2,\ldots,g_k),h^*]$,

and their duals, obtained by interchanging M and m throughout. Indeed, by (2.4) and (2.5), for all combinations of values of the variables at which f assumes the value 0, both members of each identity assume the value 1. For all combinations of values of the variables at which f assumes the value 1, both members of (6.1a) reduce to $M[\overline{g},h^*]$, both members of (6.2a) reduce to $M[M(g_1,g_2,\ldots,g_k),h^*]$, and both members of (6.3a) reduce to $M[m(g_1,g_2,\ldots,g_k),h^*]$.

In view of the arguments h^* present in these identities and the commutative nature of the M and m operators, the identities may be applied repeatedly to a given expression for the removal or insertion of redundant arguments.

As an illustration of the usefulness of (6.1a), note that

$$\begin{aligned} a \oplus b = \overline{a}b + a\overline{b} &= M[M(\overline{a},b),M(a,\overline{b})] \\ &= M[M\{M(a,b),b\},M\{a,M(a,b)\}] \end{aligned}$$

which yields the usual four NAND-element circuit for $a \oplus b$ (Figure 1.1). The dual identity (6.1b) is used to obtain the corresponding circuit for $a \odot b$.

Consider again the problem of implementing a full adder with NAND elements. We rewrite the expression

$$s = \bar{a}\,\bar{b}\,c_{in} + \bar{a}\,b\,\bar{c}_{in} + a\,\bar{b}\,\bar{c}_{in} + abc_{in}$$

for the sum digit as follows:

$$s = a\,\bar{b}\,\bar{c}_{in} + \bar{a}\,b\,\bar{c}_{in} + (\bar{a}+b)(a+\bar{b})c_{in}.$$

This yields the NAND expression

$$s = M[M(a,\bar{b},\bar{c}_{in}),M(\bar{a},b,\bar{c}_{in}),M\{M(a,\bar{b}),M(\bar{a},b),c_{in}\}].$$

By repeated use of the law of redundancy (6.1a), the preceding expansion may be altered so as to remove the complements, thus:

$$s = M[M\{a,M(a,b),M(a,c_{in})\},M\{b,M(a,b),M(b,c_{in})\},$$
$$M\{M\{a,M(a,b)\},M\{b,M(a,b)\},c_{in}\}].$$

Now, by inserting a redundant argument M(a,b) into $M(a,c_{in})$ in the first major argument and into $M(b,c_{in})$ in the second major argument, and then inserting a redundant c_{in} into each of the first two arguments of the third major argument, we get extensive repetition of the subfunctions:

$$s = M[M\{a,M(a,b),M[a,c_{in},M(a,b)]\},$$
$$M\{b,M(a,b),M[b,c_{in},M(a,b)]\},$$
$$M\{M[a,c_{in},M(a,b)],M[b,c_{in},M(a,b)],c_{in}\}].$$

Because of the repetition of subfunctions, this rather formidable appearing expression actually corresponds to a simple and advantageous circuit (Figure 6.1). The same circuit is obtained by Maley and Earle by a sequence of operations with maps.

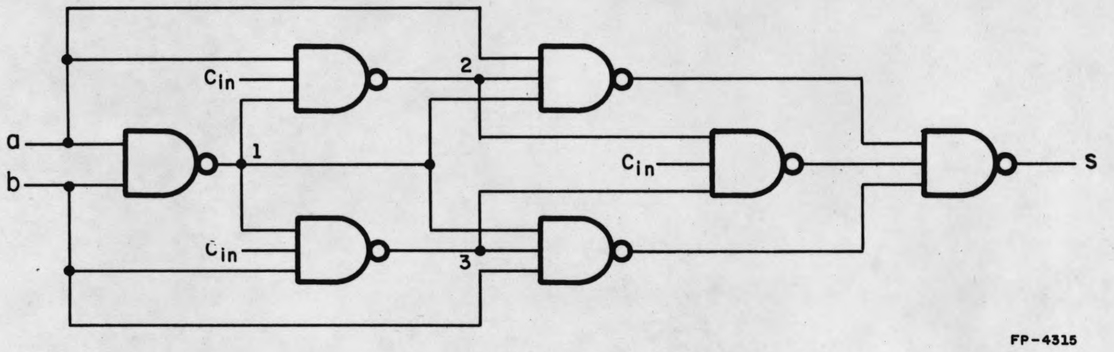


Figure 6.1. Sum Digit of Full Adder.

A major advantage of the preceding circuit appears when we implement the carry-out digit:

$$c_{out} = ab + ac_{in} + bc_{in}$$

or

$$c_{out} = M[M(a,b),M(a,c_{in}),M(b,c_{in})].$$

Inserting a redundant argument M(a,b) into each of the last two major arguments, we have

$$c_{out} = M[M(a,b),M\{a,c_{in},M(a,b)\},M\{b,c_{in},M(a,b)\}],$$

all three arguments of which have been generated in the expression for s whose circuit diagram appears in Figure 6.1, at the points labeled 1, 2, and 3, respectively. The circuit for c_{out} thus requires only a single additional NAND-gate (Figure 6.2).

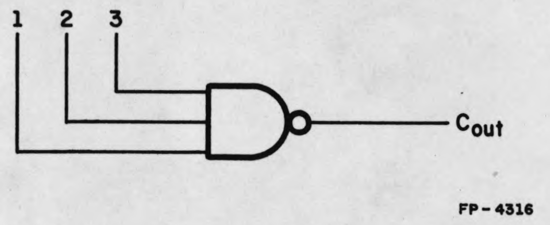


Figure 6.2. Carry-Out Digit of Full Adder.

The full adder implemented in this fashion requires a total of 23 inputs, none complemented, only 8 gates, and 5 stages.

As these examples suggest, the laws of redundancy are powerful tools for the transformation of M and m functions in ways that permit the sharing of hardware and the reduction of gate requirements.

The initial factoring of the AND-OR-NOT expression for s was not really essential, for we have the identity

$$(6.4a)\quad M[M\{M(f_1,f_2),M(\overline{f}_1,\overline{f}_2)\},h^*]$$
$$= M[M(f_1,\overline{f}_2),M(\overline{f}_1,f_2),h^*],$$

which could have accomplished the same result. This identity may be proved with the aid of (2.4) and (2.5), just as we proved (6.1a), (6.2a), and (6.3a). Details are left to the reader. Note how this identity, like each of the factoring rules, affects the number of stages, that is, the depth of a list of nested parentheses.

To apply (6.4a) to the NAND representation of s, we first use (4.2a) to factor c_{in} in the expression for s:

$$s = M[M(\bar{a},b,\bar{c}_{in}),M(a,\bar{b},\bar{c}_{in}),M\{M[M(a,b),M(\bar{a},\bar{b})],c_{in}\}].$$

Then (6.4a) gives

$$s = M[M(\bar{a},b,\bar{c}_{in}),M(a,\bar{b},\bar{c}_{in}),M\{M(a,\bar{b}),M(\bar{a},b),c_{in}\}],$$

after which the law of redundancy is applied as before.

As another example of the use of redundancy, consider the even parity function of three variables x,y,z:

$$\begin{aligned} S_{0,2}(x,y,z) &= \bar{x}\,y\,z + x\,\bar{y}\,z + x\,y\,\bar{z} + \bar{x}\,\bar{y}\,\bar{z} \\ &= (\bar{x}y + x\bar{y})z + (x+\bar{y})(\bar{x}+y)\bar{z} \\ &= M[M\{\bar{x}y + x\bar{y},z\},M\{x+\bar{y},\bar{x}+y,\bar{z}\}]. \end{aligned}$$

Now substitute for $\bar{x}y + x\bar{y}$ the expression

$$M[M\{(x,y),y\},M\{M(x,y),x\}]$$

and transform $x+\bar{y}$ and $\bar{x}+y$ into NAND functions. This yields

$$\begin{aligned} S_{0,2}(x,y,z) = M[&M(M\{M(x,y),y\},M\{M(x,y),x\}],z), \\ &M(M\{M(x,y),y\},M\{M(x,y),x\},\bar{z})]. \end{aligned}$$

Now, in the first major subargument, insert a redundant z into the minor subargument immediately preceding z and in the second major subargument, combine <u>both</u> minor arguments immediately preceding $\bar{z}$ with $\bar{z}$ in redundant fashion thus obtaining a common subargument

$$g = M(M\{M(x,y),y\},M\{M(x,y),x\},z)$$

in each of the two major arguments so

$$S_{0,2}(x,y,z) = M[M\{g,z\},M(M\{M(x,y),y\},M\{M(x,y),x\},g)].$$

These last two expressions define the five-stage circuit of Figure 6.3.

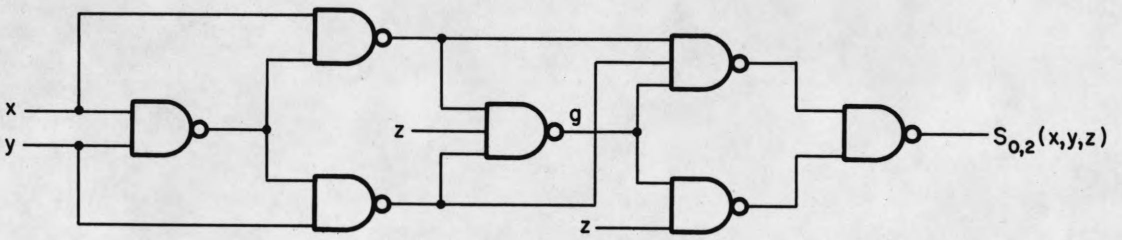
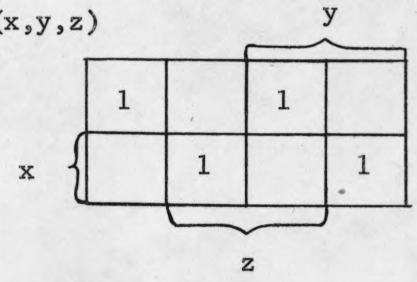


Figure 6.3. Even Parity Function

A map of $S_{0,2}(x,y,z)$



shows that

$$\bar{x}\,y\,z + x\,\bar{y}\,z + x\,y\,\bar{z} + \bar{x}\,\bar{y}\,\bar{z} = (\bar{x}+y+z)(x+\bar{y}+z)(x+y+\bar{z})(\bar{x}+\bar{y}+\bar{z}).$$

Since this is a self-dual identity, the NANDs may all be replaced by NORs in the circuit of Figure 6.3 without altering the output.

7. THE HYBRID ASSOCIATIVE LAWS

The operations M and m are not associative, that is, ordinarily

$$M[f_1, M(f_2, f_3)] \neq M[M(f_1, f_2), f_3]$$

and

$$m[f_1, m(f_2, f_3)] \neq m[m(f_1, f_2), f_3],$$

as expansion of the several members will reveal. However, some association of arguments is permissible according to the hybrid associative laws, which the reader should prove:

$$M[f_1, f_2, \ldots, f_k, g^*] = M[m(\overline{f}_1, \overline{f}_2, \ldots, \overline{f}_k), g^*] \tag{7.1a}$$

and dually. As many of these associations as are feasible may be effected.

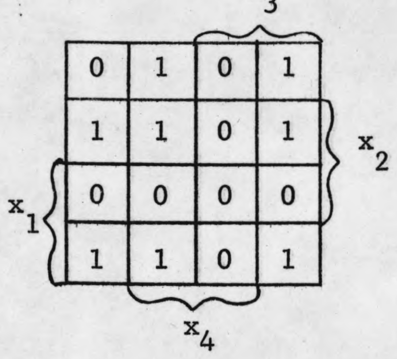
These laws are sometimes useful for removing unwanted complements and for building duplicate arguments. A simple example is this:

$$\overline{a}\,\overline{b}\,c\,d + a\,b\,\overline{c}\,\overline{d} = M[M\{m(a,b), c, d\}, M\{a, b, m(c,d)\}].$$

8. SOME MORE EXAMPLES

NAND-NOR algebra does not eliminate the need for clever Boolean algebra. The two are complementary and should be used together as effectively as possible.

Example 8.1: From the map



we have

$$f(X) = (x_1 + x_2 + x_3 + x_4)\cdot\overline{x_1x_2}\cdot\overline{x_3x_4}$$
$$= m[m(x_1,x_2,x_3,x_4), x_1x_2, x_3x_4]$$

which permits use of a standard AOI chip with expander (Figure 8.1).

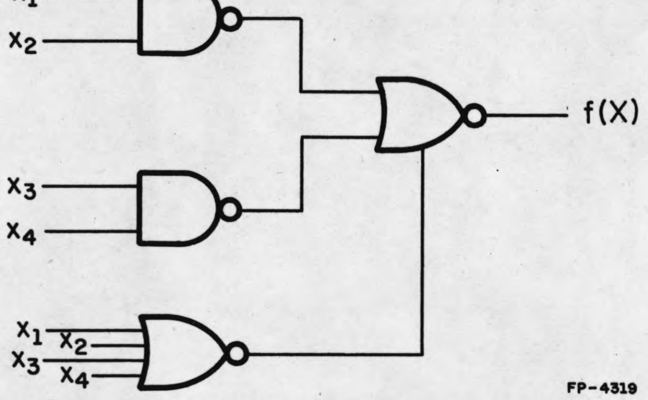
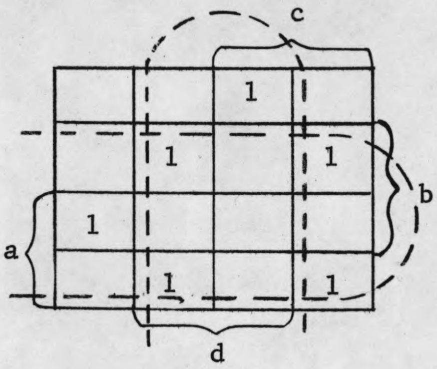
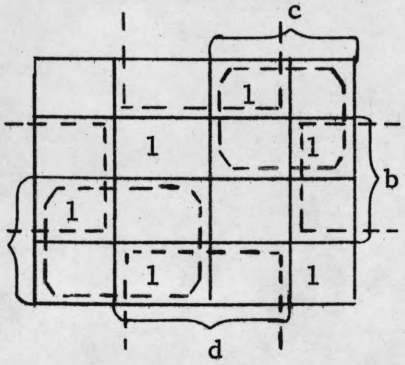


Figure 8.1

Example 8.2: Suppose it is desired to implement $S_2(a,b,c,d)$ using exclusive-ORs and NANDs. One could manipulate the algebraic expression, but it is much better to draw the map twice:





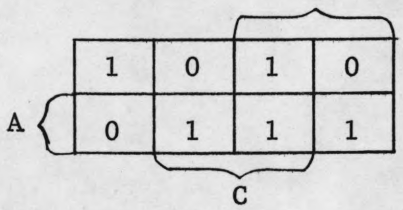
The intersections of the exclusive-OR regions show that S

$$S_2(a,b,c,d) = (a\oplus b)(c\oplus d) + (a\oplus c)(b\oplus d)$$
$$= M[M(a\oplus b, c\oplus d), M(a\oplus c, b\oplus d)].$$

Example 8.3: Consider the symmetric function

$$f(A,B,C) = AB + AC + BC + \overline{A}\,\overline{B}\,\overline{C}.$$

From the map



we see that

$$f(A,B,C) = m[m(A,B,\overline{C}), m(A,\overline{B},C), m(\overline{A},B,C)]$$

which by the laws of redundancy becomes

$$f(A,B,C) = m[m\{A,B,m(A,B,C)\},m\{A,C,m(A,B,C)\},m\{B,C,m(A,B,C)\}],$$

so the construction of duplicate arguments also removes all appearances of complemented variables (Figure 8.2).

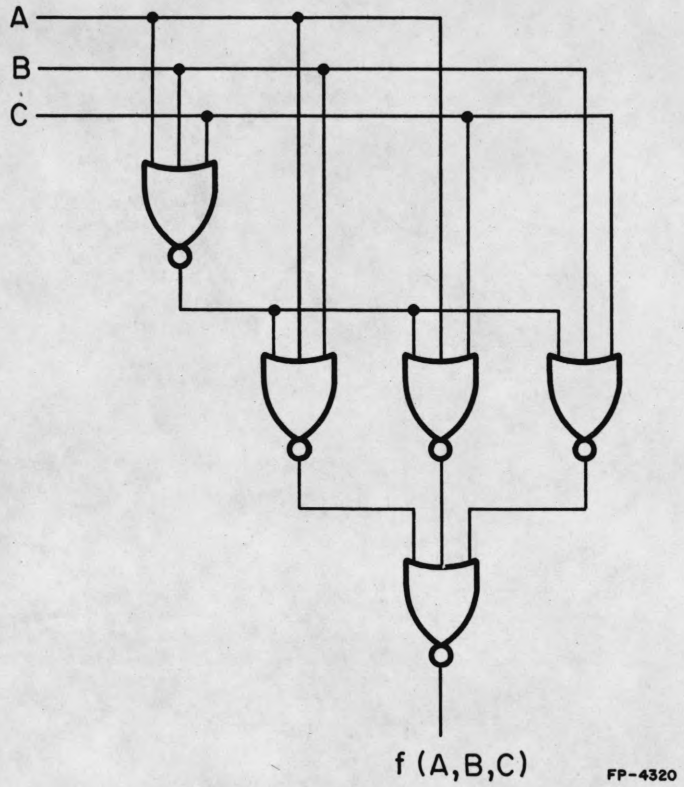


Figure 8.2

From the function as originally given, we have

$$f(A,B,C) = M[M(A,B),M(A,C),M(B,C),M(\bar{A},\bar{B},\bar{C})].$$

By the laws of redundancy we have then

$$f(A,B,C) = M[M(A,B),M(A,C),M(B,C),M\{M(A,B),\overline{M}(\bar{A},\bar{B},\bar{C})\}].$$

Since $\overline{M}(\bar{A},\bar{B},\bar{C}) = \overline{\overline{\bar{A}\,\bar{B}\,\bar{C}}} = \bar{A}\,\bar{B}\,\bar{C} = m(A,B,C)$, this becomes

$$f(A,B,C) = M[M(A,B),M(A,C),M(B,C),M\{M(A,B),m(A,B,C)\}].$$

The circuit is shown in Figure 8.3.

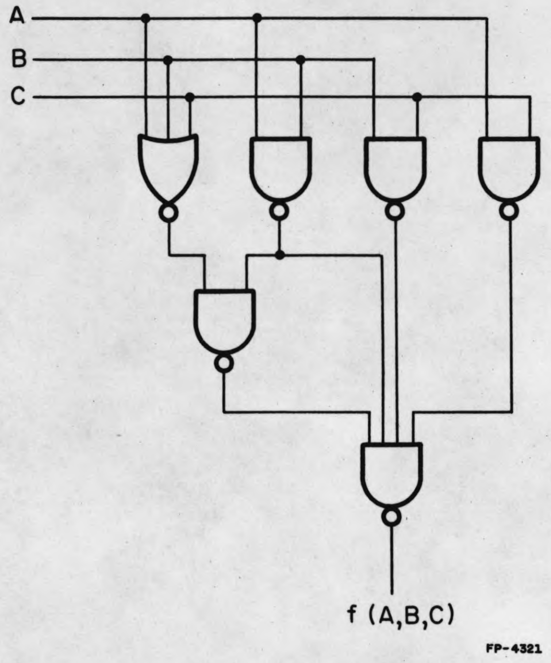


Figure 8.3

This representation has one more gate than the preceding one has, but the number of inputs is the same.

One can also note from the map that

$$\begin{aligned} f(A,B,C) &= (\overline{A}+B+C)(A+\overline{B}+C)(A+B+\overline{C}) \\ &= (\overline{A}\,\overline{B}\,\overline{C}+B+C)(A+\overline{A}\,\overline{B}\,\overline{C}+C)(A+B+\overline{A}\,\overline{B}\,\overline{C}) \end{aligned}$$

from which we have again the m-expansion of f which is represented in Figure 8.2. The point here is that the rules of NAND-NOR algebra sometimes suggest useful transformations that might be overlooked if one depended only on AND-OR-NOT manipulations (and vice versa, of course).

<u>Example 8.4</u>: Consider the symmetric function

$$S_1(a,b,c,d) = \bar{a}\,\bar{b}\,\bar{c}\,d + \bar{a}\,\bar{b}\,c\,\bar{d} + \bar{a}\,b\,\bar{c}\,\bar{d} + a\,\bar{b}\,\bar{c}\,\bar{d}$$
$$= M[M(\bar{a},\bar{b},\bar{c},d),M(\bar{a},\bar{b},c,\bar{d}),M(\bar{a},b,\bar{c},\bar{d}),M(a,\bar{b},\bar{c},\bar{d})].$$

We can apply the hybrid associative law (7.1a) to obtain

$$S_1(a,b,c,d) = M[M\{m(a,b,c),d\},M\{m(a,b,d),c\},$$
$$M\{m(a,c,d),b\},M\{m(b,c,d),a\}].$$

Alternatively, we could write

$$S_1(a,b,c,d) = \overline{a+b+c}\cdot d + \overline{a+b+d}\cdot c + \overline{a+c+d}\cdot b + \overline{b+c+d}\cdot a.$$

Then the basic transformation laws yield the same NAND-NOR expression for $S_1(a,b,c,d)$.

9. NAND-NOR ALGEBRA: THE BASIC LAWS

The operations NAND and NOR are governed by a set of identities which are closely related to the basic laws of Boolean algebra but which, because of the cumulative effect of repeated complementations, show interesting contrasts with AND-OR-NOT identities.

We now examine the basic NAND-NOR identities systematically. Those we have used in prior sections will be included at appropriate places in this list. Many of these identities will be used only rarely if AND-OR-NOT and NAND-NOR manipulations are used jointly in the natural way.

Throughout, the functions f, g, h referred to are arbitrary switching functions, expressed in any form, of a common set of variables.

Because of the way in which NAND and NOR are defined in terms of of the cummutative operations AND and OR, we have at once <u>the commutative laws</u>: If $i_1, i_2, \ldots, i_k$ is any permutation of $1, 2, \ldots, k$, we have

(9.1a) $$M(f_1, f_2, \ldots, f_k) = M(f_{i_1}, f_{i_2}, \ldots, f_{i_k}),$$

(9.1b) $$m(f_1, f_2, \ldots, f_k) = m(f_{i_1}, f_{i_2}, \ldots, f_{i_k}).$$

The operations M and m are <u>not associative</u>; that is, ordinarily

$$M[f_1, M(f_2, f_3)] \neq M[M(f_1, f_2), f_3]$$

and

$$m[f_1, m(f_2, f_3)] \neq m[m(f_1, f_2), f_3].$$

These inequalities are readily checked by transformation of their members to AND-OR-NOT form. One would expect these results since, in each case, f_1 is subjected to one complementation on the left and to two on the right. The counting of cumulative complementations is often useful in detecting errors or inequalities. However, as (6.4a) shows, when functions appear repeatedly, one can be misled by this device.

When more than one operation is involved, certain regroupings are possible, however. These are summarized in the hybrid associative laws:

(9.2a) $$M(f_1,f_2,\ldots,f_k,h^*) = M(f_1 f_2 \cdots f_k,h^*),$$

(9.2b) $$m(f_1,f_2,\ldots,f_k,h^*) = m(f_1+f_2+\cdots+f_k,h^*),$$

which are immediate from the definitions of M and m and which may also be written

(9.3a) $$M(f_1,f_2,\ldots,f_k,h^*) = M[m(\bar{f}_1,\bar{f}_2,\ldots,\bar{f}_k),h^*],$$

(9.3b) $$m(f_1,f_2,\ldots,f_k,h^*) = m[M(\bar{f}_1,\bar{f}_2,\ldots,\bar{f}_k),h^*].$$

Because of the commutative nature of M and m, these properties may be applied repeatedly and to whatever blocks of arguments are of interest. Thus if

$$\{\overline{i_1,i_2,\ldots,i_{r_1}};\overline{i_{r_1+1},i_{r_1+2},\ldots,i_{r_2}};\ldots;\overline{i_{r_p+1},i_{r_p+2},\ldots,i_k}\}$$

is any partition of $\{1,2,\ldots,k\}$, then

(9.4a) $$M(f_1,f_2,\ldots,f_k,g^*) = M[m(\bar{f}_{i_1},\ldots,\bar{f}_{i_{r_1}}),m(\bar{f}_{i_{r_1+1}},\ldots,\bar{f}_{i_{r_2}}),\ldots,m(\bar{f}_{i_{r_p+1}},\ldots,\bar{f}_{i_k}),g^*].$$

As always, the corresponding identity (9.4b) is obtained by interchanging M and m throughout in (9.4a).

The hybrid associative laws are useful for exploiting already available inputs, for controlling fan-in and fan-out, and for eliminating complements when that is desirable. The proof, by translation to AND-OR-NOT form, is left to the reader.

The operations M and m are also not distributive; that is, ordinarily M does not distribute across m:

$$M(f_1,m(f_2,f_3)) \neq m(M(f_1,f_2),M(f_1,f_3))$$

and m does not distribute across M:

$$m(f_1,M(f_2,f_3)) \neq M(m(f_1,f_2),m(f_1,f_3)).$$

Note that in each case f_1 is complemented once on the left and twice on the right.

In NAND-NOR algebra, the factoring laws (4.1) and (4.2) play the role usually played by distributive laws in the factoring process. For completeness, we include these laws in the current list. We have

$$\text{(9.5a)} \quad M[M(f,g_1),M(f,g_2),\ldots,M(f,g_p),h^*]$$
$$= M[M\{f,M(\overline{g}_1,\overline{g}_2,\ldots,\overline{g}_p)\},h^*]$$

and its generalization

$$\text{(9.6a)} \quad M[M(f^+,g_1^+),M(f^+,g_2^+),\ldots,M(f^+,g_k^+),h^*]$$
$$= M[M(f^+,M\{M(g_1^+),M(g_2^+),\ldots,M(g_k^+)\}),h^*].$$

The laws (9.5b) and (9.6b) are written in the usual way.

By the laws of idempotency for AND and OR and by the definitions (2.1), we have the laws of idempotency for NAND and NOR:

(9.7a) $$M(f,f,g^*) = M(f,g^*),$$

(9.7b) $$m(f,f,g^*) = m(f,g^*).$$

Similarly, we have immediately, from the definitions of M and m, the laws of operation with 0 and 1:

(9.8a) $$M(0,g^*) = 1,$$

(9.8b) $$m(1,g^*) = 0,$$

in which 0 and 1 are controlling inputs and

(9.9a) $$M(1,f^+) = M(f^+),$$

(9.9b) $$m(0,f^+) = m(f^+),$$

in which 0 and 1 are redundant inputs.

As our proofs of (6.1), (6.2), and (6.3) indicate, the rules (9.8) and (9.9) are particularly useful in the proof of other identities.

The laws of complementarity are also immediate from the definitions of the NAND and NOR functions:

(9.10a) $$M(f,\overline{f},g^*) = 1,$$

(9.10b) $$m(f,\overline{f},g^*) = 0.$$

Let us denote $\overline{M(f_1,f_2,\ldots,f_k)}$ by $\overline{M}(f_1,f_2,\ldots,f_k)$ and similarly for m. Then, again directly from the definitions of M and m, we have DeMorgan's laws for NAND and NOR:

(9.11a) $$\overline{M}(f_1,f_2,\ldots,f_k) = m(\overline{f}_1,\overline{f}_2,\ldots,\overline{f}_k),$$

(9.11b) $$\overline{m}(f_1,f_2,\ldots,f_k) = M(\overline{f}_1,\overline{f}_2,\ldots,\overline{f}_k).$$

In words, the complement of a NAND is the NOR of the separate complements and the complement of a NOR is the NAND of the separate complements.

The law of involution takes the forms

(9.12a) $$M(M(f)) = f,$$

(9.12b) $$m(m(f)) = f,$$

which are not essentially different since, when only one argument is involved, M and m are by definition the same function, namely, the complement. We have therefore also the mixed laws

(9.13a) $$m(M(f)) = f$$

(9.13b) $$M(m(f)) = f.$$

10. THE PRINCIPLE OF DUALITY FOR NAND AND NOR

The lists of pairs of identities in preceding sections reflect the principle of duality for NAND and NOR: Given any identity involving only the operations M and m and the constant functions 0 and 1, another valid identity is obtained by interchanging the symbols M and m and interchanging 0 and 1 throughout the given identity. This rule of duality for NAND-NOR algebra is a consequence of the fact that the definitions of M and m are dual in AND and OR.

11. ABSORPTION LAWS AND RELATED IDENTITIES

The absorption laws of Boolean algebra suggest consideration of $M(f,m(f,g))$, $M(f,m(\overline{f},g))$, and the corresponding expressions with M and m interchanged. As following identities show, familiar patterns of Boolean algebra are altered by the complementing aspects of M and m. We list only one law of each dual pair. Proofs are simply translations of AND-OR-NOT expressions to NAND-NOR form or are applications of (9.8) and (9.9), and are omitted for the most part.

We have first, in their simplest forms, <u>the reduction laws</u>,

(11.1a) $$M(f,m(f,g)) = 1,$$

<u>the laws of redundancy</u>,

(11.2a) $$M(f,m(\overline{f},g)) = M(f,m(g)) = M(f,\overline{g}),$$

and

(11.3a) $$M(f,M(f,g)) = M(f,M(g)) = M(f,\overline{g}),$$

<u>and the absorption laws</u>,

(11.4a) $$M(\overline{f},M(f,g)) = M(\overline{f}) = f.$$

It is not hard to derive <u>the generalized laws of redundancy</u>, which we have already employed in Section 6:

(11.5a) $$M[f,M(f,g^{+}),h^{*}] = M[f,M(g^{+}),h^{*}]$$

and

(11.6a) $$M[f,m(\overline{f},g^{+})] = M[f,m(g^{+}),h^{*}].$$

A special case of (11.5) permits removal of undesired complements when certain NAND-gate outputs are already available:

(11.7a) $$M[f,\overline{g},h^*] = M[f,M(f,g),h^*].$$

Similarly, a special case of (11.6) permits introduction of complements:

(11.8a) $$M[f,g,h^*] = M[f,m(\overline{f},\overline{g}),h^*].$$

Recall that the removal of undesired complements may also be accomplished by the hybrid associative laws (9.3).

The absorption laws may first be extended thus:

(11.9a) $$M[\overline{f},M(f,g^*),h^*] = M(\overline{f},h^*).$$

These are special cases of <u>the generalized absorption laws</u>:

(11.10a) $$M[M(f^+),M(f^+,g^*),h^*] = M[M(f^+),h^*].$$

One may, of course, apply (11.5)-(11.10) repeatedly to the same expression, depending on the nature of the arguments g^*, h^*.

<u>The laws of consensus</u>, namely

$$fg + \overline{f}h + gh = fg + \overline{f}h$$

and its dual translate directly into

(11.11a) $$M[M(f,g),M(\overline{f},h),M(g,h)] = M[M(f,g),M(\overline{f},h)]$$

and its dual.

Finally, we note several identities related to the Boolean reductions

$$fg + \overline{f}g = g \quad , \quad (f+g)(\overline{f}+g) = g,$$

which translate into the NAND-NOR identities

(11.12a) $$M[M(f,g),M(\overline{f},g)] = g = M[M(g)]$$

and its dual.

A first extension of these is

(11.13a) $$M[M(f,g),M(\overline{f},g),h^*] = M[M(g),h^*]$$

and its dual, which in turn suggest <u>the generalized laws of condensation</u>:

(11.14a) $$M[M(f,g^+),M(\overline{f},g^+),h^*] = M[M(g^+),h^*]$$

and its dual. Whether f assumes the value 0 or the value 1, the left member reduces to the right. Note that the argument h^* indicates the possibility of combining further pairs of terms by the same principle.

The next two identities follow easily from the laws (11.14) and (9.10):

(11.15a) $$M[M(f,g),M(f,\overline{g}),M(\overline{f},g),M(\overline{f},\overline{g})] = 1,$$

(11.15b) $$m[m(f,g),m(f,\overline{g}),m(\overline{f},g),m(\overline{f},\overline{g})] = 0.$$

Let $f_1,f_2,\ldots,f_k$ be arbitrary switching functions of n variables and let

$$F = (f_1,f_2,\ldots,f_k).$$

Also let

$$F_i = (f_1^{e_1},f_2^{e_2},\ldots,f_k^{e_k})$$

where

$$f_j^0 = \overline{f}_j \text{ and } f_j^1 = f_j$$

and where

$$i_{\text{decimal}} = (e_1e_2\ldots e_k)_{\text{binary}}.$$

Then the identities (11.15) generalize to

(11.16a) $$\underset{i=0}{\overset{2^k-1}{M}} [M(F_i)] = 1,$$

(11.16b) $$\underset{i=0}{\overset{2^k-1}{m}} [m(F_i)] = 0,$$

special cases of which are

(11.17a) $$\underset{i=0}{\overset{2^n-1}{M}} [M(p_i)] = 1,$$

(11.17b) $$\underset{i=0}{\overset{2^n-1}{m}} [m(s_i)] = 0,$$

where the p_i and the s_i are the minterms and maxterms of $x_1, x_2, \ldots, x_n$, respectively (see the next section).

The reader should provide proofs of (11.16) and (11.17).

12. NORMAL FORMS FOR SWITCHING FUNCTIONS

We consider finally the matter of normal forms of switching functions of n variables, for which purpose we need the following definitions:

$$x^0 = \overline{x},\ x^1 = x,\ 0^0 = 1,\ 0^1 = 0,\ 1^0 = 0,\ 1^1 = 1,$$

and, for $i = 0,1,2,\ldots,2^n-1$,

$$i_{dec} = (e_1e_2\ldots e_n)_{bin},$$

$$p_i = x_1^{e_1}x_2^{e_2}\cdots x_n^{e_n} \qquad (\underline{i}\text{th minterm}),$$

$$s_i = x_1^{\overline{e}_1} + x_2^{\overline{e}_2} + \cdots + x_n^{\overline{e}_n} \quad (\underline{i}\text{th maxterm}).$$

For example, if $n=6$,

$$33_{dec} = (100001)_{bin},\ 18_{dec} = (010010)_{bin},$$

$$p_{33} = x_1^1x_2^0x_3^0x_4^0x_5^0x_6^1 = x_1\overline{x}_2\overline{x}_3\overline{x}_4\overline{x}_5x_6,$$

$$s_{18} = x_1^1+x_2^0+x_3^1+x_4^1+x_5^0+x_6^1 = x_1+\overline{x}_2+x_3+x_4+\overline{x}_5+x_6.$$

We also define, for every switching function $f(x_1,x_2,\ldots,x_n)$, and for $i=0,1,2,\ldots,2^n-1$,

$$f(i) = f(e_1,e_2,\ldots,e_n)$$

where, as before, $i_{dec} = (e_1e_2\ldots e_n)_{bin}$. (Although the two functions denoted here by "f" are really not the same since they have different domains, the notation is useful and should cause no confusion.)

From these definitions it follows at once that, for $i,j = 0,1,2,\ldots,2^{n-1}$,

$$p_j(i) = \delta_{ij},\ s_j(i) = \overline{\delta}_{ij},$$

where, for the Boolean 0 and 1,

$$\delta_{ij} = 1 \text{ if } i = j \text{ but } \delta_{ij} = 0 \text{ if } i \neq j.$$

In this notation, the disjunctive and conjunctive normal forms for f are, respectively,

(12.1a) $$f(X) = \sum_{i=0}^{2^n-1} f(i)p_i$$

and

(12.1b) $$f(X) = \prod_{i=0}^{2^n-1} (f(i) + s_i).$$

From these expansions, with the aid of formulas (2.3), we now have the NAND and NOR normal forms of f(X):

(12.2a) $$f(X) = \underset{i=0}{\overset{2^n-1}{M}} (M(f(i),p_i))$$

and

(12.2b) $$f(X) = \underset{i=0}{\overset{2^n-1}{m}} (m(f(i),s_i)).$$

Now

(12.3a) $$M(p_i) = M(x_1^{e_1}, x_2^{e_2}, \ldots, x_n^{e_n}) = x_1^{\overline{e}_1} + x_2^{\overline{e}_2} + \cdots + x_n^{\overline{e}_n} = s_i$$

and

(12.3b) $$m(s_i) = m(x_1^{\overline{e}_1}, x_2^{\overline{e}_2}, \ldots, x_n^{\overline{e}_n}) = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n} = p_i.$$

Thus $M(p_i)$ is in fact a maxterm and $m(s_i)$ is in fact a minterm and this explains the M, m notation.

Let $i_1, i_2, \ldots, i_k$ be the indices of the minterms actually present in f(X) and let $j_1, j_2, \ldots, j_q$ be the indices of the maxterms actually present in f(X). Then $k+q = 2^n$, $\{i_1, i_2, \ldots, i_k\} \cap \{j_1, j_2, \ldots, j_q\} = \phi$, and, again by

(2.3), the identities (12.2) may be replaced by

(12.4a) $$f(X) = M(M(p_{i_1}),M(p_{i_2}),\ldots,M(p_{i_k}))$$

and

(12.4b) $$f(X) = m(m(s_{j_1}),m(s_{j_2}),\ldots,m(s_{j_q})).$$

Since the literals appearing in $M(p_i) = M(x_1^{e_1},x_2^{e_2},\ldots,x_n^{e_n})$ are precisely the same as those of p_i in the disjunctive normal form of $f(X)$ and the literals appearing in $m(s_i) = m(x_1^{\overline{e}_1},x_2^{\overline{e}_2},\ldots,x_n^{\overline{e}_n})$ are precisely the same as those of s_i in the conjunctive normal form of f, one can write the NAND and NOR normal forms immediately from a truth table for f. For example, from Table 12.1, we have the M-normal form

Table 12.1

i	x_1	x_2	x_3	f(i)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

$$f(X) = M[M(\overline{x}_1,\overline{x}_2,x_3),M(\overline{x}_1,x_2,\overline{x}_3),M(x_1,\overline{x}_2,x_3),M(x_1,x_2,\overline{x}_3),M(x_1,x_2,x_3)]$$

and the m-normal form

$$f(X) = m[m(x_1,x_2,x_3),m(x_1,\overline{x}_2,\overline{x}_3),m(\overline{x}_1,x_2,x_3)].$$

From (9.8) and (9.9) we have the identities

$$M(0,p_i) = 1,\ M(1,p_i) = M(p_i)$$

and

$$m(1,s_i) = 0,\ m(0,s_i) = m(s_i)$$

which,with (9.8) and (9.9), make possible the direct reduction of the identities (12.2) to the identities (12.4).

13. CONCLUSION

The functional notation for NAND and NOR and the laws governing these operations permit the economical design of logic circuits employing gates of these two kinds or these and other compatible gates. Fan-in and fan-out can be controlled if desired and, unlike map methods, the procedures are readily applicable even when the number of variables is large. Although it is possible to work exclusively with the laws of NAND-NOR algebra, this algebra is most effectively used in conjunction with the usual switching algebra.

14. EXERCISES

1. Prove that (a) $M(f_1,f_2,\ldots,f_k) = M(f_1f_2\cdots f_i,f_{i+1}f_{i+2}\cdots f_k)$

$$= M(f_1f_2\cdots f_k),$$

(b) $M[m(\overline{a},\overline{c}),m(\overline{b},\overline{c})] = M(a,b,c)$.

2. Simplify $m[m(f,g),m(f,\overline{g}),m(\overline{f},\overline{g})]$, using only the rules of NAND-NOR algebra.

3. By transforming each member to AND-OR-NOT form, prove that

$$M[M\{M(\overline{A},\overline{B}),C\},M\{A,D,M(A,C)\},M(A,D)]$$
$$= M[M(A,C),M(B,C),M(A,D,\overline{C})].$$

4. Use the law of redundancy to remove all complements from

$$M[a,\overline{b},M(\overline{c},\overline{d})].$$

5. Use NAND-NOR algebra only to prove that

$$M[M(x_1,x_2),M\{M(x_1,x_3),x_2\}] = x_2.$$

6. Under what conditions on the variables a, b, c is it true that

(a) $M[a,M(b,c)] = M[M(a,b),c]$,

(b) $m[M(a,b),M(b,c),M(c,a)] = M[m(a,b),m(b,c),m(c,a)]$?

7. Simplify the circuit of Figure 14.1 using NAND-NOR algebra so far as possible:

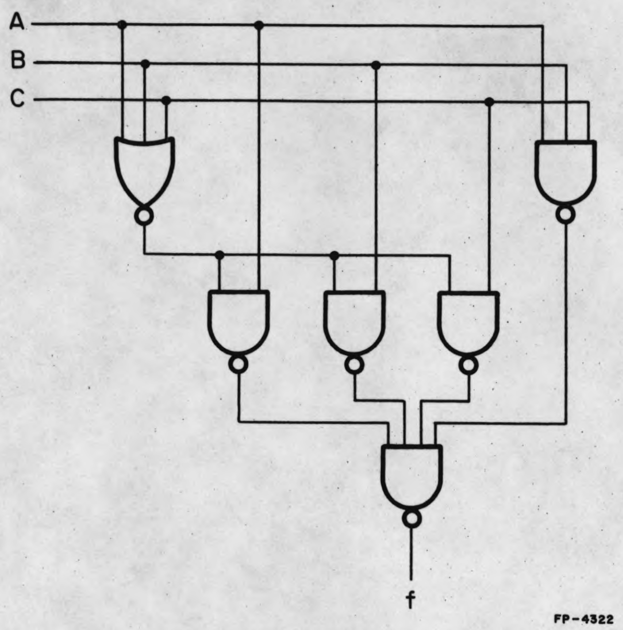


Figure 14.1

8. What function is implemented by the circuit of Figure 14.2?

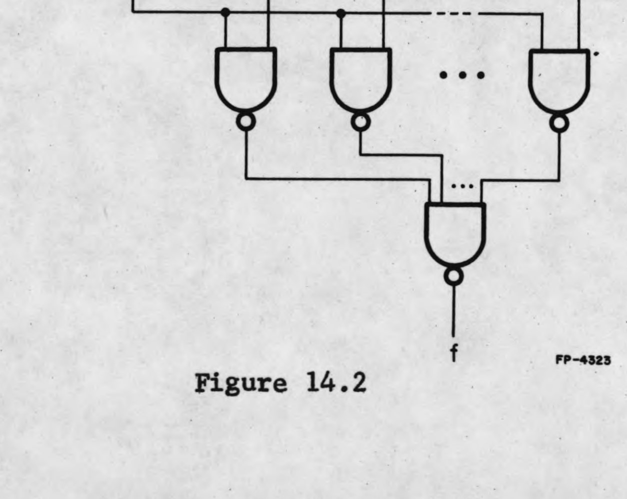


Figure 14.2

9. Find minimum-gate, NAND-circuit implementations of each of the following functions, assuming complements of inputs are available:

   (a) $f_1(A,B,C) = \bar{A}\,\bar{B} + \bar{A}\,\bar{C} + AC$,

   (b) $f_2(x_1,x_2,x_3) = x_1x_3 + x_2\bar{x}_3$,

   (c) $f_3(x_1,x_2,x_3,x_4) = \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_3 + \bar{x}_2\bar{x}_3 + \bar{x}_2\bar{x}_4$,

   (d) $S_3(x_1,x_2,x_3,x_4)$,

   (e) $g(A,B,C,D) = (A+\bar{B})(B+\bar{C})(C+\bar{D})(D+\bar{A})$,

   (f) $h(a,b,c) = ab + \bar{b}c + \bar{a}c$.

10. Find minimum-gate, NOR-circuit implementations of each of the following functions, assuming complements of inputs are available:

   (a) $f_1(A,B,C) = (A+B)(\bar{A}+C)(\bar{B}+\bar{C})$,

   (b) $f_2(x_1,x_2,x_3) = (x_1+\bar{x}_3)(x_2+x_3)$,

   (c) $f_3(a,b) = ab + \bar{a}\,\bar{b}$,

   (d) $g(x_1,x_2,x_3,x_4) = (x_1+\bar{x}_2)(x_2+\bar{x}_3)(x_3+\bar{x}_4)(x_4+\bar{x}_1)$,

   (e) $h(A,B,C) = AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC$,

   (f) $S_3(x_1,x_2,x_3,x_4)$.

11. For each of the functions of Exercises 9 and 10 find the minimum-gate NAND-NOR circuit, assuming complements of inputs are <u>not</u> available.

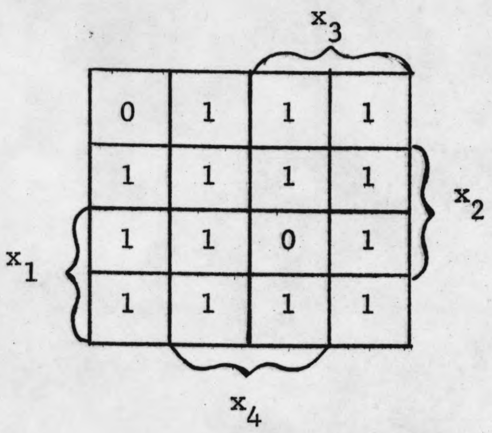
12. Use the table and maps of Section 4 to design a NOR-gate full adder with a minimum number of gates. The ideas of Section 6 may be helpful here.

13. Every identity

$$f_1 + f_2 + \cdots + f_p = g_1g_2\cdots g_q$$

defines a NAND-to-NOR (and a NOR-to-NAND) transformation. Illustrate with several examples.

14. Obtain the most economical circuit, using arbitrary chips, for the function defined by this map:



0	1	1	1
1	1	1	1
1	1	0	1
1	1	1	1

15. Use the hybrid associative law to obtain the minimum-gate NAND-NOR implementation of $S_2(A,B,C,D)$ assuming fan-in and fan-out are both restricted to 3 or less and assuming complements are not available.

16. Reduce $M[M\{M(f_1,f_2,\ldots,f_k),M(\bar{f}_1,\bar{f}_2,\ldots,\bar{f}_k)\},h^*]$ to a two-stage expression.

17. Show that the inputs to the circuit of Figure 14.3 may be permuted arbitrarily.

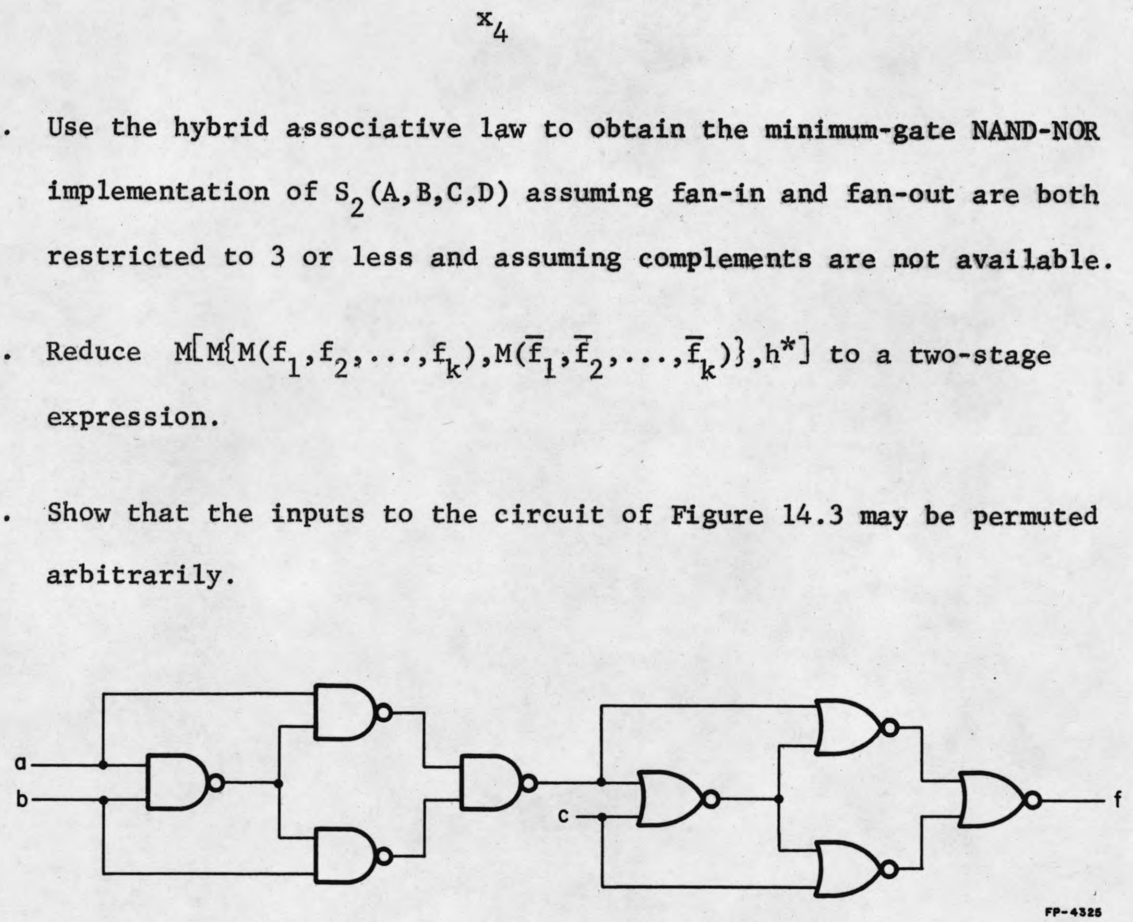


Figure 14.3

18. Under what conditions is

$$M(f_1,f_2,\ldots,f_k)\cdot M(g_1,g_2,\ldots,g_\ell) = 0?$$

19. Redesign the circuit of Figure 14.4

(a) as a 3-level NAND circuit,

(b) as a 3-level NAND-NOR circuit,

assuming complements of inputs are not available and using no more gates than five in either case.

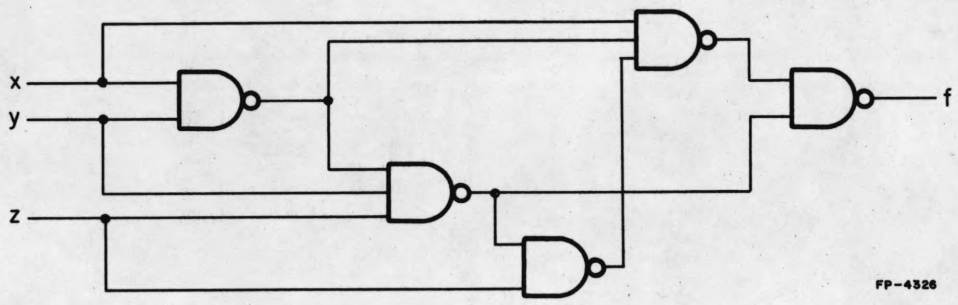


Figure 14.4