An algorithm for the synthesis of NAND logic networks using a diagrammatic approach

Donald Robert Nelson

AN ALGORITHM FOR THE SYNTHESIS OF NAND LOGIC NETWORKS USING A DIAGRAMMATIC APPROACH

BY

DONALD ROBERT NELSON

A

THESIS

submitted to the faculty of the

UNIVERSITY OF MISSOURI AT ROLLA

in partial fulfillment of the requirements of the

Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Rolla, Missouri

1966

Approved by

[illegible] (Advisor) [illegible]

[illegible] [illegible]

ABSTRACT

A diagrammatic approach is presented for the synthesis of multilevel NAND networks realizing combinational logic expressions. The network synthesized is restricted to only uncomplemented inputs. The synthesis algorithm involves the determination of minimum sum of products and product of sums expressions for a Boolean function, construction of an α-β diagram from these expressions followed by implementation with NAND gates directly from the diagram. The resulting network is a minimal or near minimal NAND gate realization of the given function. The algorithm is applicable to completely or incompletely specified Boolean functions and is extended to include NOR synthesis.

ACKNOWLEDGEMENT

The author would like to express his appreciation for the encouragement and guidance he received from his advisor, Dr. James H. Tracey.

TABLE OF CONTENTS

ABSTRACT 2

ACKNOWLEDGEMENT 3

LIST OF FIGURES 5

CHAPTER I. INTRODUCTION 6

CHAPTER II. THE SYNTHESIS OF NAND NETWORKS USING α-β DIAGRAMS 12

A. Construction of the α-β diagram 13

1. The α-set 14
2. The β-set 17
3. The α-β Diagram 19
4. Simplification of the α-β Diagram 22

B. The Network Synthesis 23

1. The Synthesis 24
2. Extension to NOR Synthesis 44
3. Worked Examples 44

CHAPTER III. SUMMARY 52

BIBLIOGRAPHY 53

VITA 54

LIST OF FIGURES

Figure		page
1	An α-β diagram illustration	12
2	Example function for the α-set determination	14
3	Example function for the β-set determination	18
4	An α-β diagram	20
5	Dual and complement α-β diagrams	21
6	Illustration of algorithm step 2	25
7	Illustration of algorithm step 4	26
8	Illustration of algorithm step 6	27
9	Illustration of algorithm step 8	28
10	Illustration of algorithm step 10	29
11	Illustration of algorithm step 12	30
12	Redundant gate removal	31
13	Illustration of rule 1	32
14	Illustration of rule 2	33
15	Two-level equivalents	34
16	Illustration of reading the α-β diagram	37
17	Modification of Figure 16	37
18	Decomposition of a diagram	43

CHAPTER I

INTRODUCTION

The problem in logical design of synthesizing NAND networks with only uncomplemented inputs available has been and still is of great importance. The NAND element is chosen because of its inherent amplification and drive capability and because it is a universal logic element. That is to say, any Boolean function may be realized using combinations of only these gates. Another reason for choosing the NAND element is the growing importance of integrated circuits in the digital field and the relative ease of fabricating these elements using integrated circuit techniques.

The networks synthesized by the method described in this paper will be restricted to only uncomplemented inputs. Frequently the inputs to the networks to be synthesized will be the outputs from other NAND networks. The complemented outputs of these gates are generally not available. It is well known that if the theory of two-stage networks using AND and OR gates is extended to this problem, the resulting networks will not necessarily be minimal or least-cost networks. In this paper, a network will be said to be minimal if it realizes the desired Boolean expression with a minimum number of inputs to a minimum number of gates.

Smith (1)* assumed the availability of both complemented and uncomplemented inputs in his synthesis procedure. A com-

*Numbers in parentheses are references to the Bibliography.

puter was utilized to investigate all of the possible ways of interconnecting a given number of NAND elements and all of the functions of three variables that each connection could generate. The minimum network was then selected. This procedure could become overwhelming for a greater number of variables.

The recent work by Dietmeyer and Schneider (2) was to develop a programmable algorithm for the synthesis of NAND and NOR networks. Again, both complemented and uncomplemented inputs were assumed available. The algorithm was developed so as to satisfy fan-in requirements of the gating used, by factoring in a prescribed manner. The object of this work is not to guarantee minimal networks but to provide speed and accuracy of design satisfying fan-in restriction.

The two papers already cited considered problems where both complemented and uncomplemented inputs were assumed. In this paper, the inputs are restricted to only uncomplemented variables. The papers reviewed below consider this latter class of problems.

For functions of three variables the work done by Hellerman (3) is significant. An exhaustive method was employed to select the minimum NAND and NOR network for generating functions of three variables with uncomplemented inputs available. All possible combinational networks with seven and fewer blocks (2^{42}) were examined (fan-in was limited to three variables). The networks were examined in ascending order of number of blocks with all combinations of inputs. The first network found to generate one of the 256 Boolean functions of three variables was selected as were all others using the same number of gates. The one with the fewest inputs was then selected as the minimal network realizing that function. This was possible as all others to be obtained would contain a larger number of gates. When it is desired to synthesize functions of more than three variables, a complete enumeration and selection techniques become unwieldy and impractical if not impossible by known techniques and computing capability. This is easily seen by noting that there are $2^{2^4} = 65,536$ functions of four variables not to mention the number of possible NAND implementations for each one.

Maley and Earle (4) present an algebraic approach to the synthesis of functions of several variables using NAND gates. This approach is to factor a sum of products expression of the given switching function in a prescribed manner. With the judicious use of added redundancy, a logically equivalent expression is obtained which is suitable for synthesis with NANDs. The complexity of the network realized using this approach depends on the facility of the user in manipulating the Boolean expression. For a large number of variables, this algebraic approach can become quite inefficient with the possibility that the resulting network has more gates and inputs than the two-level network for the minimum sum of products expression with single-input gates used as inverters.

In the previously cited work by Maley and Earle, a method of factoring on a Karnaugh map is presented for the synthesis of multi-level NAND networks. This method is easily applicable to functions of only a few variables. However, it has been this author's experience that networks synthesized by this map factoring approach tend to have excessive levels of logic. As with the previously mentioned algebraic method, the complexity of the network realized depends to a large extent on the skill of the user.

NAND networks limited to three levels with only uncomplemented inputs available have been called TANT (Three-level AND - NOT network with True inputs) networks by McCluskey (5) and Gimpel (6). Gimpel and McCluskey consider the same class of functions to be considered in this paper. However, their solutions are limited to TANT networks. Gimpel describes an approach using prime implicants of the TANT expression which are analogous but not equivalent to the prime implicants in AND - OR synthesis. The approach is similar to the Quine-McCluskey algorithm for two-level AND - OR synthesis. The resulting network is a minimum gate count TANT network. The algorithm of this paper does not restrict the solution to a TANT network.

Ellis (7) has developed a systematic procedure for synthesizing NOR and NAND networks limited to three levels. The procedure is essentially an extension of the algebraic approach of Maley and Earle utilizing redundancy and factoring so that the outputs of third-level gates may be shared by second-level gates. In this method, the prime implicants are listed and then grouped into one of three basic patterns. This grouping leads to possible reduction in the final NAND network by the sharing of gates on the input level of logic. These gates generate the negated variables in the prime implicants.

This paper presents an approach to the synthesis of multi-level NAND networks which is a modification of the work done by Akers (8). In his paper, Akers utilizes the concept of an α-β diagram to develop a method for multi-level AND - OR synthesis. The approach is diagrammatic, i. e., the network is synthesized directly from a diagram which represents the switching function. This paper is concerned with the construction of a modified α-β diagram suitable for NAND networks, diagram simplification, and diagram transformation into minimal or near-minimal NAND networks. The techniques developed by Akers for the manipulation of the diagram applicable to the problem being considered in this paper will be summarized when needed. It is the author's feeling that the method herein described is easier to apply than those in the previously cited literature.

This diagrammatic approach has the advantage of giving a visual interpretation to the concepts of fan-in, fan-out and levels of logic. In most instances, the network synthesized by this method will be minimal. Finally, the procedure will be extended to include synthesis with NOR gates through the use of the duality property of the NAND and NOR.

CHAPTER II

THE SYNTHESIS OF NAND NETWORKS USING α-β DIAGRAMS

As in Akers' method, the synthesis procedure to be presented in this chapter involves the construction and use of a diagram derived from the specifications of a switching function. A switching function $F(x_1, x_2, \ldots, x_n)$ is described by Akers (9) as an (n + 1)-column truth table defining for each n-bit binary input combination the corresponding value of F (either 0 or 1). The table must be consistent which means that F cannot be both 0 and 1 for the same input combination. In his paper, Akers utilizes the concept of a logically passive function described in a previous paper (9) to obtain what he calls α and β sets. Stated simply, any switching function realizable with only AND and OR gates is logically passive. A diagram is constructed from these α and β sets. This diagram is called the α-β diagram. The diagram has the property that reading horizontally corresponds to logical multiplication and reading vertically to logical addition. This is illustrated in Figure 1.

A	B	A	B
C	C	D	D

$$f = ABAB + CCDD = AB + CD$$
$$= (A+C)(B+C)(A+D)(B+D)$$

Figure 1. An α-β diagram illustration

The method described by Akers for the construction of this diagram may be quite lengthy and is not particularly suited for our problem. For this reason, a different method of obtaining the diagram will be presented. This method has the advantage that it involves the familiar procedure of selecting minimal sets of prime implicants and implicates from a Karnaugh map.

A. Construction of the α-β Diagram

The usefulness of the synthesis method presented in this paper is based on the relative ease with which it may be applied. The construction of the α-β diagram is an integral part of this procedure. It has been the author's experience that the method presented by Akers for this construction is somewhat tedious and lengthy. Therefore, a new and simpler method will be developed in this paper.

The α-β diagram is formed as a rectangular array having one row for each term in a sum of products representation of the switching function to be synthesized and one column for each factor in a product of sums representation for the same function. Since there may be more than one sum of products and product of sums representation of a given function, the possibility of more than one different α-β diagram representing the same function is not obviated. Because of this, the question is raised as to whether or not one diagram may be better than another in so far as synthesis is concerned. This is indeed the case. It is necessary then to select the "best" α-β diagram which will in turn yield a minimum or near minimum network. The next two sections describe how to develop this "best" α-β diagram.

1. The α-set

The terms that comprise a sum of products expression for a function, f, will be called the α-set for that function. Clearly, there may be more than one α-set for any given function. The problem is to obtain an α-set that will include all of the minimal sum of products expressions for f. The selection of such an α-set is best illustrated by an example. Consider the switching function, f, in Figure 2(a).

($f = \sum 2, 4, 5, 6, 8, 10, 11, 13, 15$)

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	1	0
0	0	1	0	1
0	1	0	0	1
0	1	0	1	1
0	1	1	1	0
0	1	1	0	1
1	1	0	0	0
1	1	0	1	1
1	1	1	1	1
1	1	1	0	0
1	0	0	0	1
1	0	0	1	0
1	1	1	1	1
1	0	1	0	1

(a)

AB \ CD	00	01	11	10
00	0	0	0	1
01	1	1	0	1
11	0	1	1	0
10	1	0	1	1

(b)

Figure 2. Example function for the α-set determination

The function is first mapped on a Karnaugh map as shown in Figure 2(b). Then all minimum sum of products expressions are formed. Minimal expressions are of interest because they yield minimum row α-β diagrams. It will be shown that simplification of the α-β diagram in terms of row and column elimination corresponds to simplification of the resulting NAND network.

For the example function there are three minimum sum of products expressions.

$$f_1 = A\bar{B}\bar{D} + \bar{A}C\bar{D} + \bar{A}B\bar{C} + ACD + ABD$$
$$f_2 = A\bar{B}\bar{D} + \bar{A}C\bar{D} + \bar{A}B\bar{C} + ACD + B\bar{C}D$$
$$f_3 = A\bar{B}\bar{D} + \bar{A}C\bar{D} + \bar{A}B\bar{C} + A\bar{B}C + ABD$$

It appears that a decision must be madeas to which minimal expression to use. This decision will be postponed by constructing a sum of products expression for the function, and corresponding α-set which includes all minimal forms. The α-β diagram will then contain a collection of all minimal sum of products of the function. Simplification of the α-β diagram to be described later corresponds to selection of the best minimal form for NAND implementation. It is a relatively easy matter to perform this selection on the α-β diagram as opposed to selection before diagramming. This points out an important advantage of the diagrammatic approach over the previously cited algebraic approaches. The minimum expression which has this property can be obtained by first logically multiplying together all minimal sum of products expressions. Thensimplify with only the following three theorems:

1. $a \cdot \overline{a} = 0$
2. $a \cdot a = a$
3. $a + ab = a$

An equivalent procedure, and one that will save time and effort in most cases is to first select the terms that appear in <u>all</u> minimum sum of products expressions for f. Designate the logical sum of these terms by r. The sums of the remaining terms in $f_1, f_2, f_3 \ldots f_n$ are designated by $s_1, s_2, s_3 \ldots s_n$. That is to say

$$f_1 = r + s_1$$
$$f_2 = r + s_2$$
$$f_3 = r + s_3$$
$$\vdots$$
$$f_n = r + s_n$$

where $f_1, f_2, f_3 \ldots f_n$ are the minimal sum of products expressions for f and therefore are all logically equivalent to f.

It then follows that

$$\begin{aligned}(f_1)(f_2)(f_3) \ldots (f_n) &= (r+s_1)(r+s_2)(r+s_3) \ldots (r+s_n) \\ &= r + s_1\, s_2\, s_3 \ldots s_n \\ &= f_\alpha\end{aligned}$$

where f_α is the desired sum of products expression. It is obvious that f_α is logically equivalent to f.

For the example function

$r = A\overline{B}\overline{D} + \overline{A}C\overline{D} + \overline{A}B\overline{C}$

$s_1 = ACD + ABD$

$s_2 = ACD + B\overline{C}D$

$s_3 = A\overline{B}C + ABD$

Therefore

$$f_\alpha = A\overline{B}\overline{D} + \overline{A}C\overline{D} + \overline{A}B\overline{C} + A\overline{B}CD + ABCD + AB\overline{C}D$$

and the α-set is

$$\{A\overline{B}\overline{D},\ \overline{A}C\overline{D},\ \overline{A}B\overline{C},\ A\overline{B}CD,\ ABCD,\ AB\overline{C}D\}$$

A summary of the steps involved in obtaining the α-set is as follows:

1. Map the function.
2. Read all minimal sum of products expressions.
3. Logically multiply together all minimal sum of products expressions.
4. Simplify the resulting sum of products expression by using only selected theorems.
5. Select all resulting terms as elements of the α-set.

2. The β-set

The set of terms from the dual of a product of sums expression for a function will be called the β-set of that function. Again, there may be more than one β-set for any given function. The β-set will be obtained in a manner similar to that used for obtaining the α-set and will utilize the same Karnaugh map. The determination of the β-set for the first example is trivial. Therefore, tc better illustrate the selection of the β-set consider the new function that is shown mapped in Figure 3.

AB \ CD	00	01	11	10
00	1	1	0	1
01	1	1	1	1
11	0	1	0	1
10	0	1	0	0

Figure 3. Example function for the β-set determination

All minimum product of sums expressions are formed. Minimal expressions are of interest because they yield minimum column α-β diagrams. As stated in connection with the α-set determination, simplification of the α-β diagram in terms of column elimination corresponds to simplification of the resulting NAND network. For this example there are two minimal product of sums:

$$f_1 = (\overline{A} + \overline{C} + \overline{D})(B + \overline{C} + \overline{D})(\overline{A} + C + D)(\overline{A} + B + D)$$

$$f_2 = (\overline{A} + \overline{C} + \overline{D})(B + \overline{C} + \overline{D})(\overline{A} + C + D)(\overline{A} + B + \overline{C})$$

The β-set is determined from the dual expressions which are as follows,

$$f_{1\ \text{dual}} = \overline{A}\overline{C}\overline{D} + B\overline{C}\overline{D} + \overline{A}CD + \overline{A}BD$$

$$f_{2\ \text{dual}} = \overline{A}\overline{C}\overline{D} + B\overline{C}\overline{D} + \overline{A}CD + \overline{A}B\overline{C}$$

The same procedure as was used in the determination of the α-set is now followed. All of these dual expressions are logically multiplied together. Simplification of the resulting expression is accomplished by the application of only the three Boolean theorems used in α-set simplification.

The terms of the resulting expression, $f_{\beta\ \mathrm{dual}}$, constitute the β-set. In the present example,

$$f_{\beta\ \mathrm{dual}} = \overline{A}\overline{C}\overline{D} + B\overline{C}\overline{D} + \overline{A}CD + \overline{A}B\overline{C}D$$

Note that by taking the dual of $f_{\beta\ \mathrm{dual}}$, a product of sums expression is obtained, f_β, which is equivalent to f.

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

The terms of $f_{\beta\ \mathrm{dual}}$ form the elements of the β-set. The β-set for the example is

$$\left\{\overline{A}\overline{C}\overline{D},\ B\overline{C}\overline{D},\ \overline{A}CD,\ \overline{A}B\overline{C}D\right\}.$$

Therefore, the selection of which minimal product of sums expression to use is postponed to the α-β diagram simplification where it can be done with greater facility.

3. The α-β Diagram

After the α and β sets have been determined, the α-β diagram can be constructed. This diagram is formed as a rectangular array having one row for each term of the α-set and one column for each term of the β-set. The literals in each term of the α-set are the labels for the rows. The literals of each term of the β-set are the labels for the columns. In square i,j are entered the literals that appear as labels for both row i and column j. Once the literals have been entered in the squares, the labels on the rows and columns may be omitted. The diagram constructed in this manner will be called the α-β diagram.

The α-set for the function of Figure 2(a) was determined as

$\{A\bar{B}\bar{D},\ \bar{A}C\bar{D},\ \bar{A}B\bar{C},\ A\bar{B}CD,\ ABCD,\ AB\bar{C}D\}$

The β-set is easily determined as

$\{\bar{A}\bar{B}D,\ BC\bar{D},\ A\bar{C}\bar{D},\ ABC\}$.

The α-β diagram for this function is shown with the row and column labels retained for clarity in Figure 4.

	$\bar{A}\bar{B}D$	$BC\bar{D}$	$A\bar{C}\bar{D}$	ABC
$A\bar{B}\bar{D}$	$\bar{B}$	$\bar{D}$	$A\bar{D}$	A
$\bar{A}C\bar{D}$	$\bar{A}$	$C\bar{D}$	$\bar{D}$	C
$\bar{A}B\bar{C}$	$\bar{A}$	B	$\bar{C}$	B
$A\bar{B}CD$	$\bar{B}D$	C	A	AC
$ABCD$	D	BC	A	ABC
$AB\bar{C}D$	D	B	$A\bar{C}$	AB

Figure 4. An α-β diagram

From the preceeding example it is seen that the resulting α-β diagram may contain squares with more than one entry. The α-β diagram to be used in the synthesis algorithm must consist of squares with single entries. Before this problem is discussed, it will be advantageous to make the following definitions.

Definition 1.

A column (row) of an α-β diagram that has as entries only complemented variables will be referred to as a <u>complemented column</u> (row).

Definition 2.

A column (row) of an α-β diagram that has as entries only uncomplemented variables will be referred to as an uncomplemented column (row).

In the synthesis procedure it will be necessary to be able to form the α-β diagram of the dual and of the complement of a function if the α-β diagram of the function itself has already been determined. Since the dual of an expression is obtained by interchanging all occurrences of + and $\cdot$, and of 1 and 0, the α-β diagram of the dual of a function is obtained by simply interchanging the elements of the α and β sets. This procedure is illustrated in Figure 5(a). By DeMorgan's Law, the α-β diagram of the complemented function is obtained by interchanging the α and β sets and complementing all literals. This is shown in Figure 5(b).

A	$\overline{B}$	D	D
B	C	$\overline{A}$	B
$\overline{D}$	A	$\overline{B}$	C

α-β diagram of f

A	B	$\overline{D}$
$\overline{B}$	C	A
D	$\overline{A}$	$\overline{B}$
D	B	C

α-β diagram of the dual of f

(a)

$\overline{A}$	$\overline{B}$	D
B	$\overline{C}$	$\overline{A}$
$\overline{D}$	A	B
$\overline{D}$	$\overline{B}$	$\overline{C}$

(b) α-β diagram of $\overline{f}$

Figure 5. Dual and complement α-β diagram

In this paper the different letters in a Boolean expression used to denote statements concerning bivalued situations will be called variables. For example, in the expression

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

there are five variables A, B, C, D, and E. Each occurance of a variable or its complement is called a literal. There are seven literals in the example expression. In light of this definition, the literal $\overline{B}$ is not a variable but is the complement of a variable, namely B. Unless otherwise specified, the terms variable and uncomplemented variable will be used interchangeably.

4. Simplification of the α-β Diagram

The α-β diagram to be used in the algorithm must consist of squares with a single entry. Akers (8) lists the following four rules which may be applied in simplifying the α-β diagram:

1. Rows and columns may be rearranged in any order. This is possible because no particular order is required in labeling the rows and columns with elements of the α-β sets.
2. If one row (column) has the same literals (and perhaps others) in the same squares as a second, it may be removed.
3. A literal may be removed from any row (column) in which it does not appear alone. This must be done one at a time.
4. All but one literal may be removed from any square.

When the α-β diagram is being simplified, i.e., reduced to a diagram consisting of squares with single entries, the literals that are retained in given squares will be selected so as to yield α-β diagrams with the following desired properties. The list is in descending order of priority.

1. Remove redundant literals so as to yield a diagram that contains only uncomplemented variables.
2. Remove redundant literals so as to yield a row whose entries are a single complemented variable.
3. Remove redundant literals so as to yield a diagram that contains only complemented variables.
4. Remove redundant literals so as to yield a diagram with all columns complemented or uncomplemented.
5. Remove redundant literals so as to form an uncomplemented row.
6. Remove redundant literals so as to form a maximum number of complemented columns. Remaining columns should show a minimum number of complemented variables.

After elimination of redundant literals according to rules 2 or 5, reconsider the α-β diagram exclusive of the complemented or uncomplemented row.

B. The Network Synthesis

After the α-β diagram is simplified, the synthesis algorithm will be employed to obtain the required network realization. The words "gate" and "NAND gate" will be used interchangeably in this algorithm.

The concept of a level or stage of gating used in the synthesis algorithm will be as described by Gimpel (6). A gate in a network is said to be a first-stage gate (or to be on level one) if it is an output gate. A gate is said to be an n^{th}-stage gate (or on level n) if it feeds only an $(n-1)^{th}$-stage gate (or gates). If a network is loop free, i.e. free of feedback loops, then such a numbering scheme is well defined. The networks synthesized by the following algorithm will be loop free.

So as not to disrupt the flow of the algorithm from step to step, the algorithm will simply be stated. Justification for the steps will follow.

1. The Synthesis Algorithm

Step 1.

Does the α-β diagram contain only uncomplemented variables? If it does, go to step 2. If it does not, go to step 3.

Step 2.

Synthesize the function that is diagrammed, f, by using two levels of logic. The variables on each row form the inputs to second-level gates, one gate for each row. The outputs of these gates are the inputs to a first-level gate. The output of this gate is f. This procedure is illustrated in Figure 6.

A	B	C
D	E	F
G	H	I

α-β diagram satisfying conditions of step 2. The variables A, B, C, etc. are generalized and all functions of the same variables.

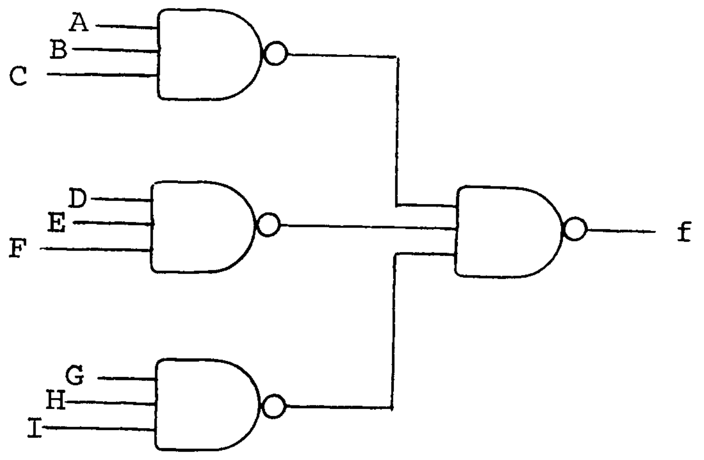


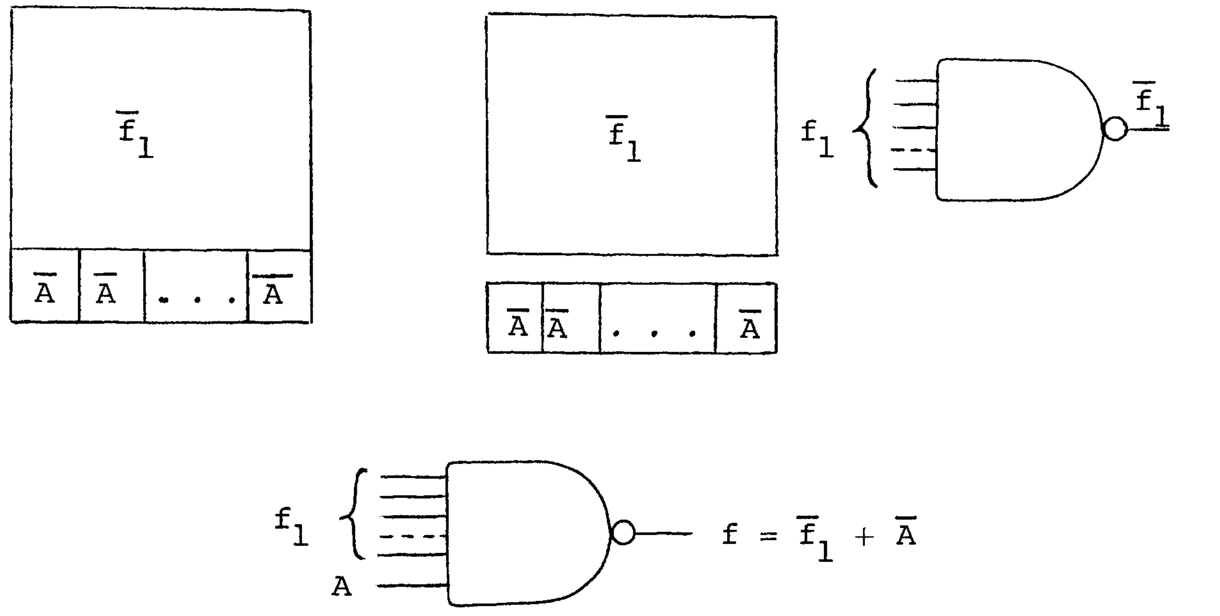
Figure 6. Illustration of algorithm step 2

Step 3.

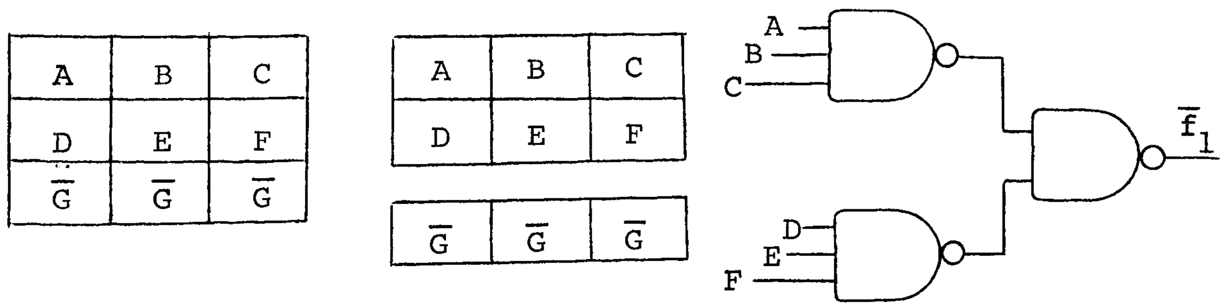
Does the α-β diagram contain a row with a single complemented variable? If it does, go to step 4. If not, go to step 5.

Step 4.

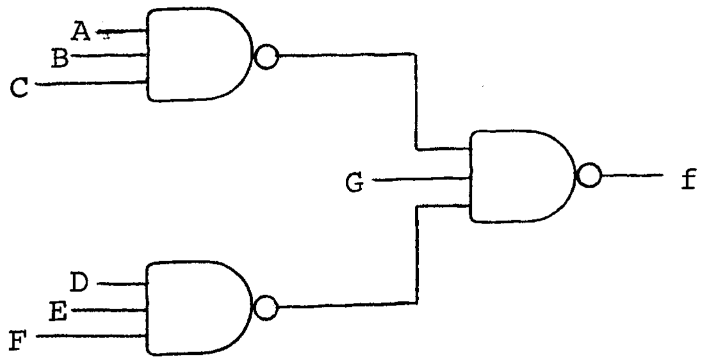
Remove from the α-β diagram that row whose entries are all the same complemented variable, say $\bar{A}$. The new α-β diagram represents $\bar{f}_1$ where $f = \bar{f}_1 + \bar{A}$. $\bar{f}_1$ is then synthesized by applying the algorithm to its α-β diagram starting with step 1. The variable A is input to the output gate for $\bar{f}_1$. With the addition of the A input the output of this gate is now the desired function, f. The general procedure is illustrated in Figure 7(a) with a specific example in Figure 7(b).



(a)



and finally



(b)

Figure 7. Illustration of algorithm step 4

<u>Step 5</u>.

Does the α-β diagram contain only complemented variables? If it does, go to step 6. If not go to step 7.

<u>Step 6</u>.

Form the complement of the α-β diagram, $\overline{\alpha-\beta}$. Synthesize $\overline{\alpha-\beta}$ with two levels of logic as in step 2. The output of the network obtained is $\bar{f}$. The desired function, f, is obtained by using $\bar{f}$ as the input to a single-input gate. The output of this gate is f. This step is illustrated in Figure 8.

$\bar{A}$	$\bar{D}$
$\bar{B}$	$\bar{E}$
$\bar{C}$	$\bar{F}$

α-β diagram for f

A	B	C
D	E	F

$\overline{\alpha-\beta}$

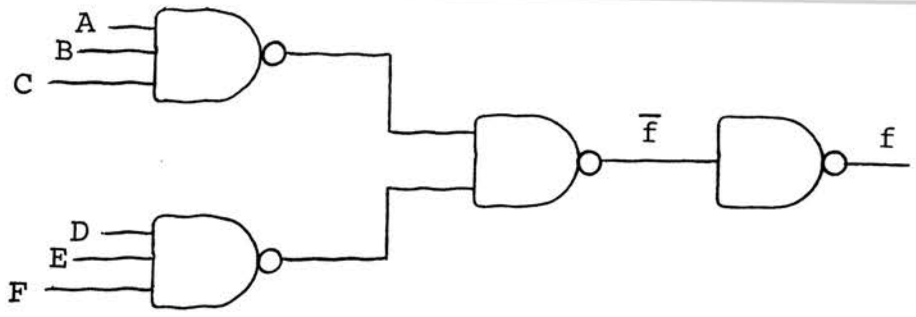


Figure 8. Illustration of algorithm step 6

<u>Step 7</u>.

Does the α-β diagram consist of only complemented and uncomplemented columns? If it does, go to step 8. If not, go to step 9.

Step 8.

Synthesize the function, f, with a TANT network. The variables in each complemented column are the inputs to third-level gates, one gate for each column. The outputs of all third-level gates are inputs to all second-level gates, one for each row, along with the uncomplemented variables in each row. The outputs of the second-level gates are inputs to the first-level gate. The output of the first-level gate is f. This is illustrated in Figure 9.

A	A	B	$\overline{C}$	$\overline{D}$
D	C	D	$\overline{A}$	$\overline{A}$
D	B	D	$\overline{A}$	$\overline{A}$
A	A	D	$\overline{C}$	$\overline{B}$

α-β diagram of f

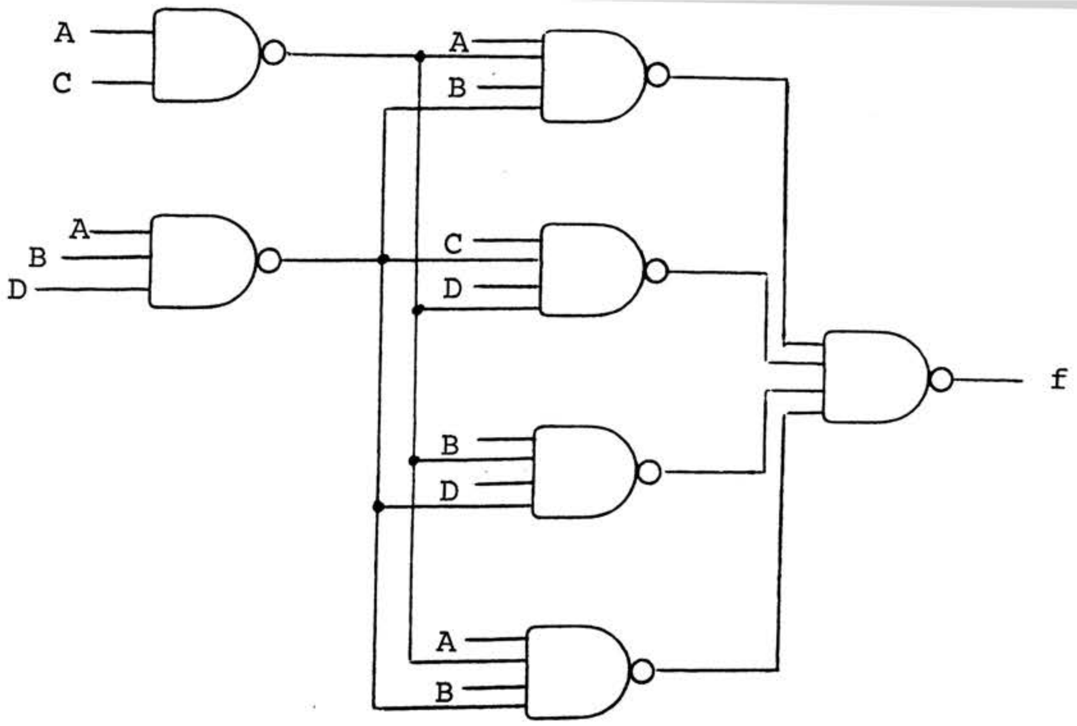


Figure 9. Illustration of algorithm step 8

Step 9.

Does the α-β diagram contain an uncomplemented row? If it does, go to step 10. If not, go to step 11.

Step 10.

The uncomplemented row in the α-β diagram for the function, f, is removed. The removed row represents $\overline{f}_2$ and the remaining α-β diagram represents $\overline{f}_1$ where $f = \overline{f}_1 + \overline{f}_2$. The diagram of $\overline{f}_1$ is then synthesized by applying the algorithm to its α-β diagram starting with step 3. The variables in the uncomplemented row are the inputs to a single gate. The output of this gate, which is f_2, is then input to what was the output gate of $\overline{f}_1$. The output of this gate is now the desired function f. This procedure is illustrated in Figure 10.

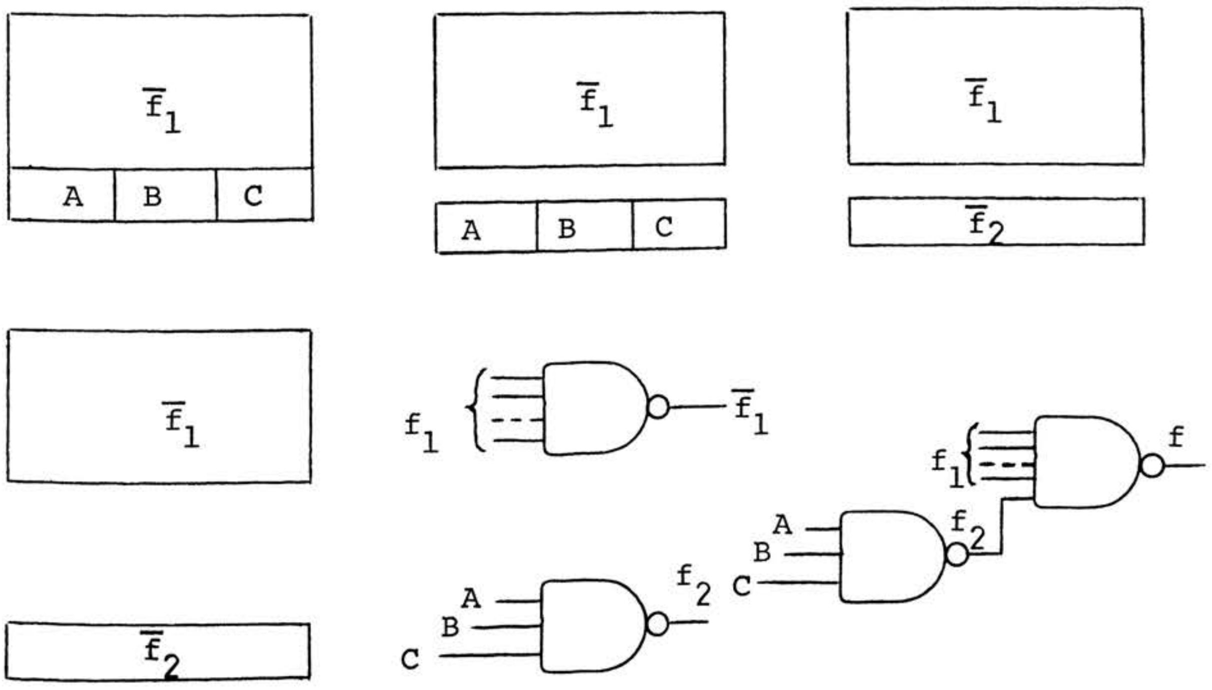


Figure 10. Illustration of algorithm step 10

Step 11.

Does the α-β diagram have at least one complemented column? If it does, go to step 12. If not, go to step 13.

Step 12.

The variables in each complemented column are the inputs to third-level gates, one gate for each complemented column. The outputs of all third-level gates are inputs to all second-level gates. There is one second-level gate for each row. The literals in each row exclusive of those in the complemented columns are inputs to the second-level gates. If a literal in a complemented column is repeated in its row, it will be input to the second-level gate for that row. If a literal input at the second stage is a complemented variable, it will be formed by using a single-input gate as an inverter. The outputs of the second-level gates are the inputs to the first-level gate. The output of this gate is the desired function f. This step is illustrated in Figure 11.

$\bar{B}$	$\bar{A}$
A	$\bar{C}$

$\Rightarrow$

$\bar{B}$
A

$\bar{A}$
$\bar{C}$

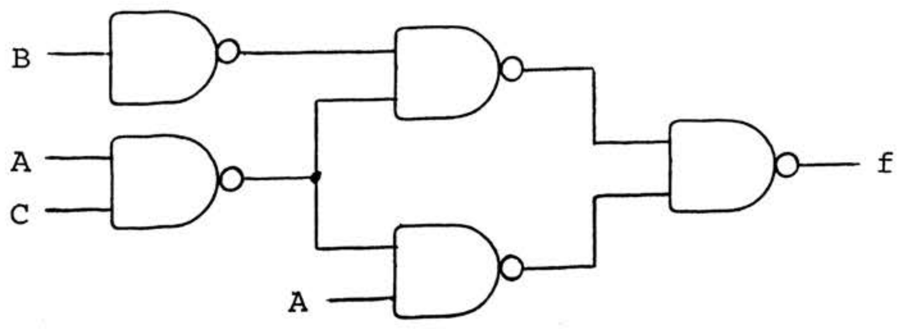


Figure 11. Illustration of algorithm step 12

<u>Step 13</u>.

It is well known that two levels of NAND logic, or NAND - NAND networks, are equivalent from terminal considerations to AND - OR logic. Select the minimal sum of products expression for the function that contains the fewest complemented variables. Synthesize the function with two levels of NAND circuitry as if synthesizing with AND - OR logic. If complemented variables are needed they will be obtained by using a single-input gate as an inverter on the third level.

A network synthesized by formal application of the algorithm may contain gates which feed the same gates and have identical inputs. All but one of these gates are redundant and may be removed. This simplification is so obvious that such gates would normally not be generated even though specified by the algorithm. This procedure is illustrated in Figure 12.

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

α-β diagram for f

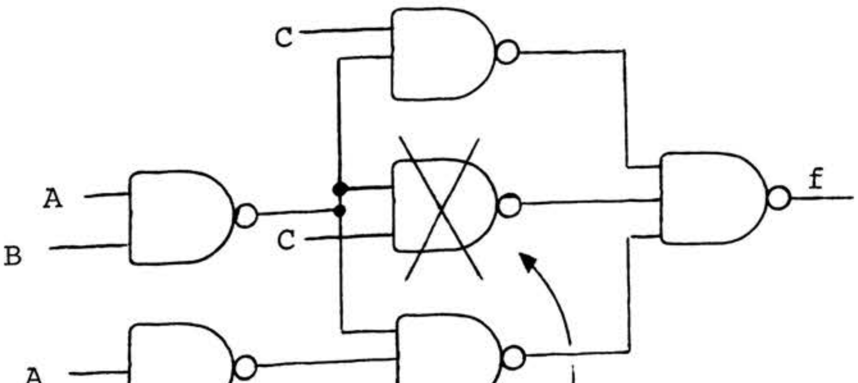


Figure 12. Redundant gate removal

After the network has been realized, it may be possible to reduce the number of inputs. The following two rules will be helpful in the reduction.

Rule 1. Consider a gate, N_1, that has as inputs the set of inputs $\{x\} = x_1, x_2, \ldots x_n$ and the outputs of at least two gates, N_2 and N_3. Also assume that $\{x\}$ is a set of inputs to N_2 along with the set of inputs $\{y\} = y_1, y_2, \ldots y_n$, none of whose elements are included in $\{x\}$. Further assume at least one of the inputs $\{y\}$, y_1, is an input to N_3. The input to N_1 from N_3 is redundant and may be removed. This is illustrated for the general case in Figure 13(a). A more specific example is shown in Figure 13(b).

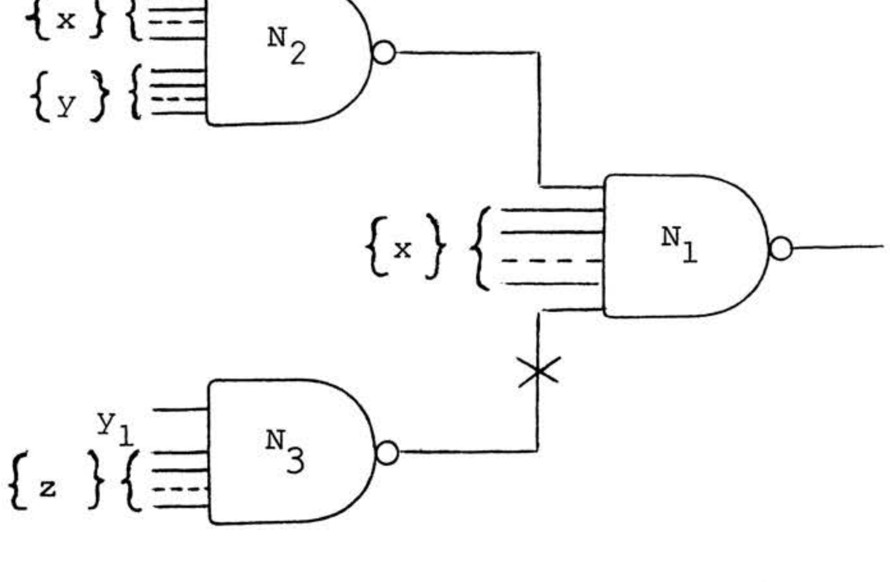


Figure 13(a). Illustration of rule 1

Inputs marked by an $\times$ are redundant and may be removed.

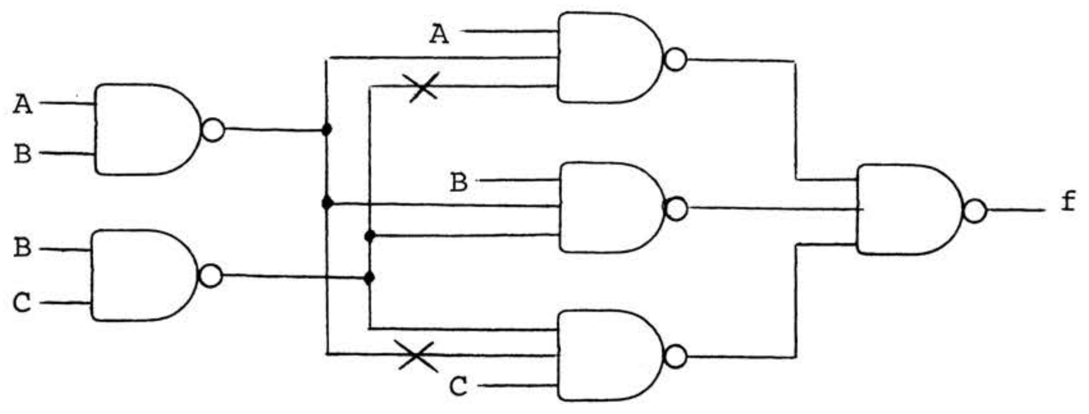
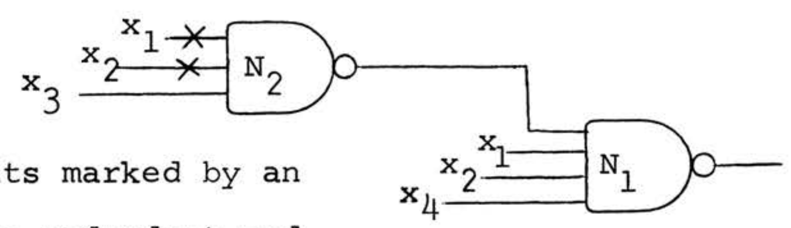


Figure 13(b). Illustration of rule 1

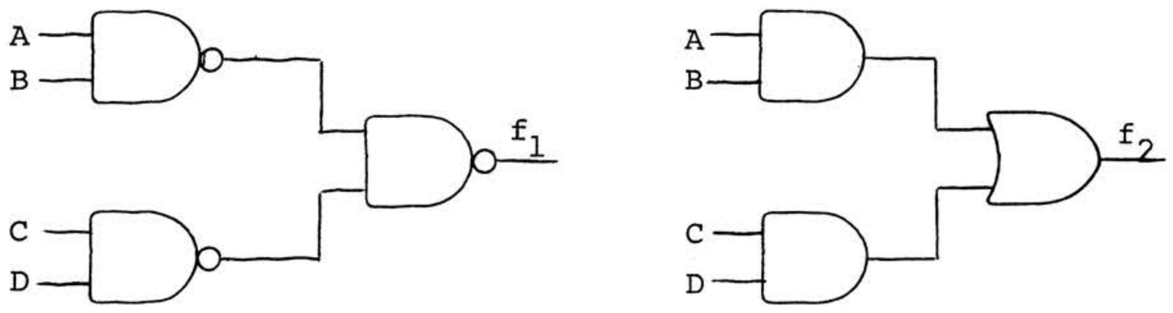
Rule 2. Consider a gate, N_1, that has among its inputs the output of a gate, N_2. Suppose further that N_2 feeds only N_1 and that N_1 and N_2 have some common inputs, x_1, x_2, . . . x_n. These inputs are redundant inputs to N_2 and may be removed. This is illustrated in Figure 14.



Inputs marked by an $\times$ are redundant and may be removed.

Figure 14. Illustration of rule 2

The following discussion is intended to support the synthesis algorithm and network simplification rules. As previously stated, it is well known that two levels of NAND logic is equivalent from a terminal point of view to two levels of AND - OR logic. This equivalent property is illustrated in Figure 15.



$f_1 = \overline{\overline{AB}\ \overline{CD}} = AB + CD$ $\qquad\qquad$ $f_2 = AB + CD$

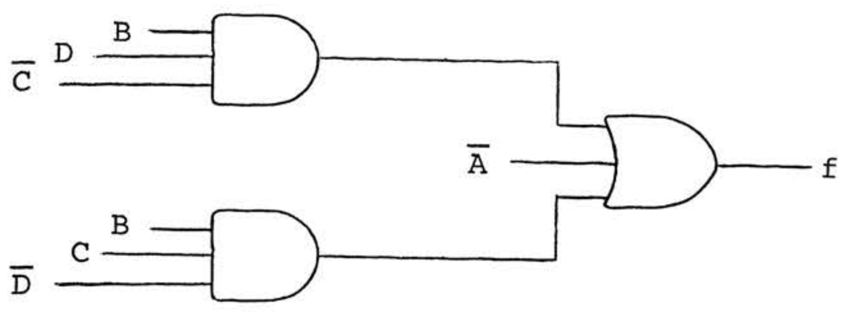
(a) $\qquad\qquad$ (b)

Figure 15. Two-level equivalents

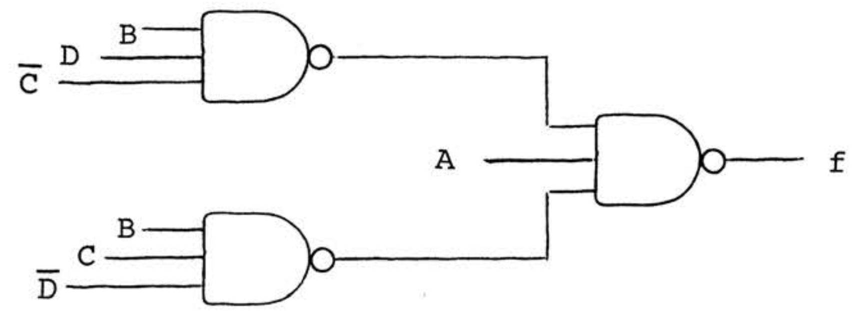
In general, if Boolean equations are arranged in a form where the output gate is an OR (sum of products of sums of products, etc.), they may be implemented in NAND logic simply by changing all gates in the AND - OR implementation to NANDs and complementing all single variable inputs to odd levels. To illustrate this consider the Boolean function

$$f = \overline{A} + B\overline{C}D + BC\overline{D}$$

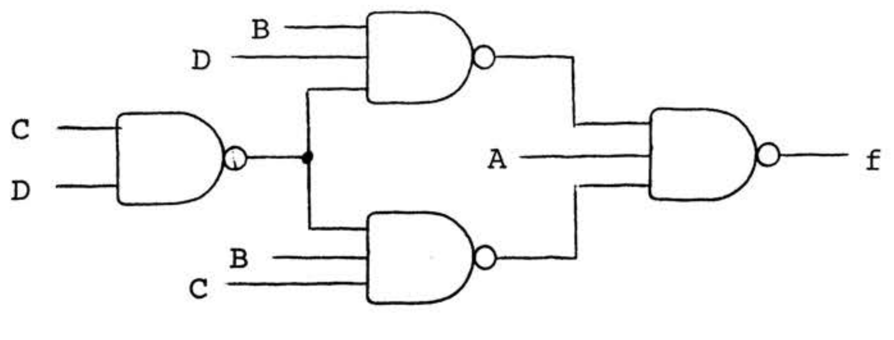
The AND - OR network is



and the equivalent NAND representation is



If complemented variables are not available, as is the assumption in this paper, single-input NANDs would be used to form $\overline{C}$ and $\overline{D}$ to give a total of five gates. If redundancy is added and f is factored as $\overline{A} + BD(\overline{C} + \overline{D}) + BC(\overline{C} + \overline{D})$ a savings of gates may be obtained by using a single gate to generate the common factor. A NAND network using only four gates that represents this factored expression is

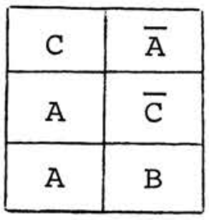


It is clear from the previous discussion that if a Boolean function is to be implemented in NAND circuitry and only uncomplemented inputs are available, any variable that appears complemented in the function must be an input to an odd-level gate. Further, from an extension of the two-level NAND and AND - OR equivalence, it is obvious that odd-level NAND gates perform an equivalent OR operation with single inputs complemented and even-level gates the AND operation.

From the foregoing illustration and discussion one can deduce that when implementing a Boolean function in NAND logic, it is desirable to arrange the function in a form where the output gate is an OR with all occurrences of complemented variables appearing singly in a sum. Complemented variables appearing singly in a sum corresponds to the variable itself as an input to an equivalent OR, or odd-level gate, in the NAND realization. Also, the judicious addition of redundancy and factoring in such a manner that common factors are obtained corresponds to the sharing of gates in the network realization.

The addition of redundancy and proper factoring of a function to produce a minimum or near minimum NAND network is easily implemented through use of the α-β diagram.

Akers has shown that reading the α-β diagram horizontally corresponds to logical multiplication and that reading vertically corresponds to logical addition. A sum of products expression may be read by forming the logical sum of the rows. Likewise, a product of sums expression may be read by forming the logical product of the columns. An illustration of this is shown in Figure 16.



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

Figure 16. Illustration of reading the α-β diagram

Multilevel expressions may be obtained by combinations of the above two procedures. Thus, multilevel forms of the example are:

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

and

$$f = (\overline{A} + \overline{C})(A + C) + AB$$

Akers has also shown that one may let a section of an α-β diagram be represented by a new variable. Consider such a modification of Figure 16 shown in Figure 17.

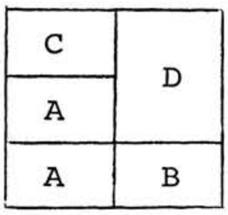


Figure 17. Modification of Figure 16

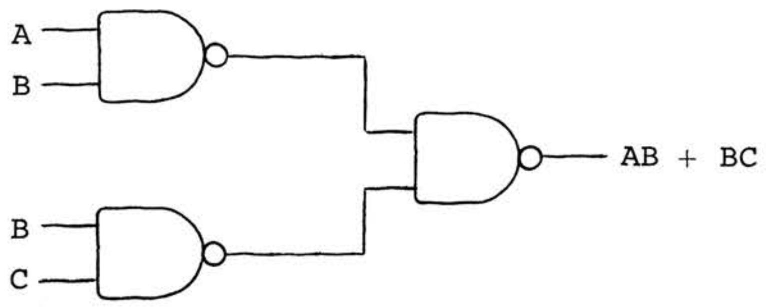
One method for reading this diagram yields $f = CD + AD + AB$ where $D = \overline{A} + \overline{C}$. Therefore $f = C(\overline{A} + \overline{C}) + A(\overline{A} + C) + AB$.

With the preceeding discussion in mind, attention will now be focused on the individual steps of the algorithm. An α-β diagram which satisfies the conditions of each step will be shown. The algebraic expression will be read and the resulting network indicated.

An α-β diagram satisfying the condition of step 1 is

A	B
C	B

The two-level sum of products expression for this diagram is read as AB + BC. By application of step 2 the NAND network is

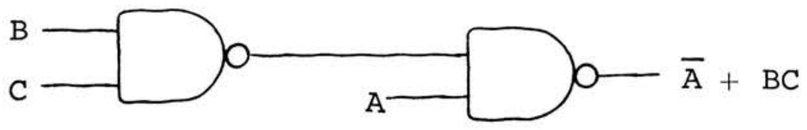


whose output is the desired function as indicated.

An α-β diagram satisfying the condition of step 3 is

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

The sum of products expression for this diagram is $\overline{A}$ + BC and by application of step 4 the required NAND realization is



Clearly, the removal of a row with a single complemented variable is equivalent to forcing this complemented variable to appear singly in the sum. The simplest implementation will show this variable as an input to the output gate. The conditions of step 5 are satisfied by the following α-β diagram. The complement α-β diagram, $\overline{\alpha-\beta}$, is also shown.

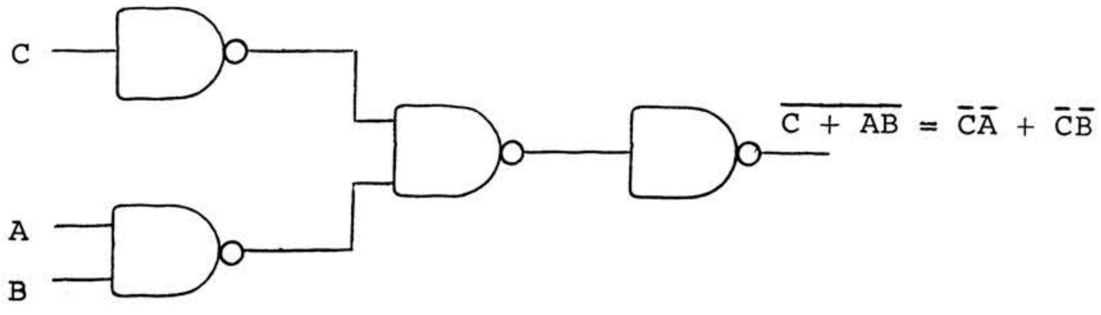
$\bar{B}$	$\bar{C}$
$\bar{A}$	$\bar{C}$

α-β

B	A
C	C

$\overline{\alpha-\beta}$

The complement of the function is read from $\overline{\alpha-\beta}$ as C + AB. By step 6 the desired function is synthesized by the following network,

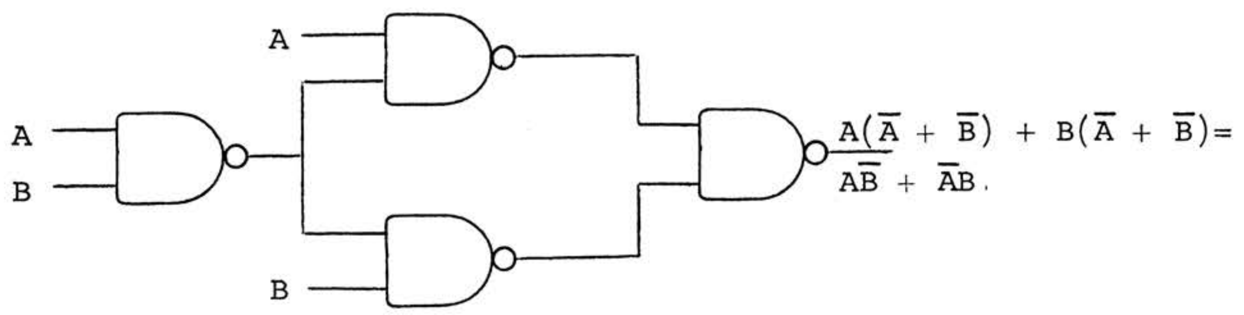


the output of which is the function read from the α-β diagram. An α-β diagram containing all complemented variables usually requires a large number of inverters at the input. Complementation of the α-β diagram has the effect of performing this complementation with a single inverter on the output.

As an example of an α-β diagram that fulfills the condition of step 7, consider that of the EXCLUSIVE-OR function $A\bar{B} + \bar{A}B$ shown below. The breakdown of the diagram by step 8 is also shown to indicate how the procedure of removing a complemented column is equivalent to the addition of redundancy so that a gate may be shared.

$$\begin{array}{|c|c|}\hline A & \overline{B} \\ \hline B & \overline{A} \\ \hline \end{array} \Rightarrow \begin{array}{|c|}\hline A \\ \hline B \\ \hline \end{array} \quad \begin{array}{|c|}\hline \overline{B} \\ \hline \overline{A} \\ \hline \end{array} \Rightarrow \begin{array}{|c|}\hline A \\ \hline B \\ \hline \end{array} \quad \begin{array}{|c|}\hline \overline{B} \\ \hline \overline{A} \\ \hline \end{array}$$

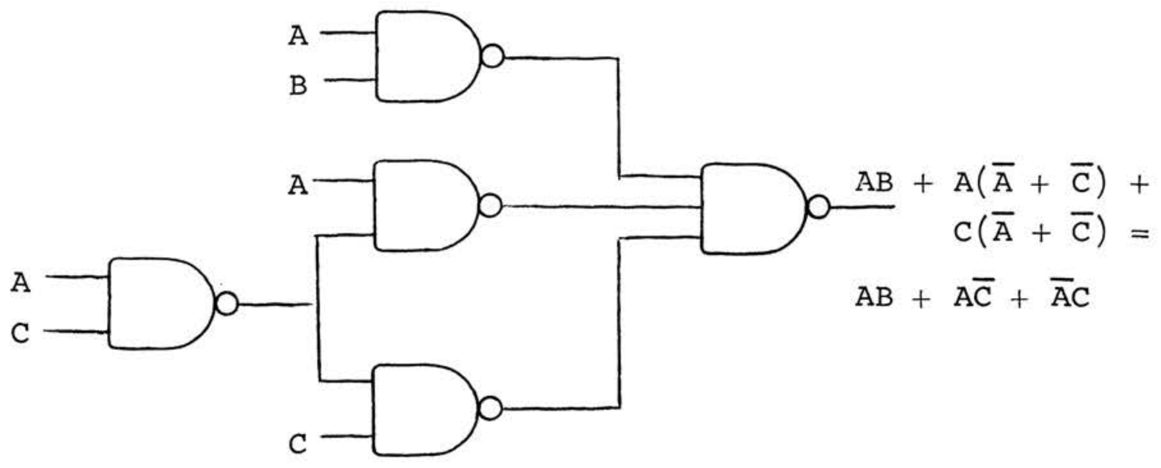
The expression read in this manner is $A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B})$ and the resulting NAND network is


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

The conditions of step 9 are satisfied by the next α-β diagram.

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

The sum of products expression is $AB + \overline{A}C + A\overline{C}$. Implementation by step 10 provides a gate for the term AB on the second level as shown in the resulting network below.

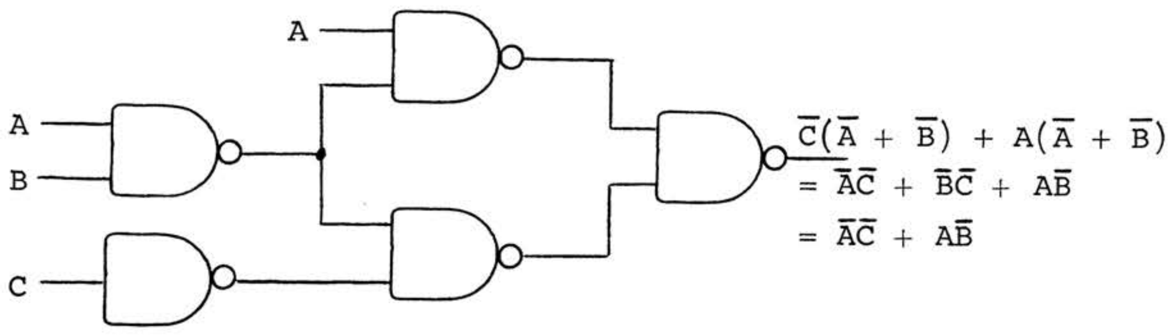

A
B
A
A
C
C
$AB + A(\overline{A} + \overline{C}) +$
$C(\overline{A} + \overline{C}) =$
$AB + A\overline{C} + \overline{A}C$

Prime implicants with no complemented variables have been termed "frontal prime implicants" by McCluskey (5). The variables of such terms will appear as inputs to a second-level gate in the minimum network realization.

An α-β diagram which satisfies the condition of step 11 is shown below.

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

The multilevel expression $A(\overline{B} + \overline{A}) + \overline{C}(\overline{A} + \overline{B})$ may be read so that the third-level gate generating $\overline{A} + \overline{B}$ may be shared. The network which results by application of algorithm step 12 is


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

The reading of complemented columns in this manner allows for sharing of gates on the third level and reduces the number of gates that must be used as inverters to generate complemented variables.

Support will now be given to the input simplification rules. The function realized by the network of Figure 13(b) before application of rule 1 is

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

After application of rule 1, the function realized is

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

It can easily be shown that the two expressions are logically equivalent. Application of rule 1 is an application of the Boolean simplification theorem

$$X(\overline{X} + \overline{Y})(\overline{Y} + \overline{Z}) = X(\overline{X} + \overline{Y})$$

The output of the network in Figure 14 before the application of input reduction rule 2 is

$$\overline{x}_1 + \overline{x}_2 + \overline{x}_4 + x_1x_2x_3$$

It is clear that x_1 and x_2 are redundant in the term $x_1x_2x_3$.

Application of rule 2 removes these redundant literals by application of the Boolean identity

$$X + \overline{X}Y = X + Y.$$

In some cases when a large α-β diagram is encountered, it may be advantageous to decompose the diagram into smaller diagrams, each corresponding to a subfunction of the original Boolean expression. This will be most desirable with diagrams that are to be synthesized by application of step 13, which results in a two-level network plus inverters on some inputs.

A description of the process of decomposition using α-β diagrams is given by Akers (1). Given the diagram of a function F, cut the α-β diagram into smaller α-β diagrams $\overline{G}$ and $\overline{H}$. $\overline{H}$ and $\overline{G}$ may now be synthesized independently. A network realization of F may be obtained by combining the outputs of the networks realizing G and H in a manner similar to that shown in Figure 10. This procedure is illustrated in Figure 18.

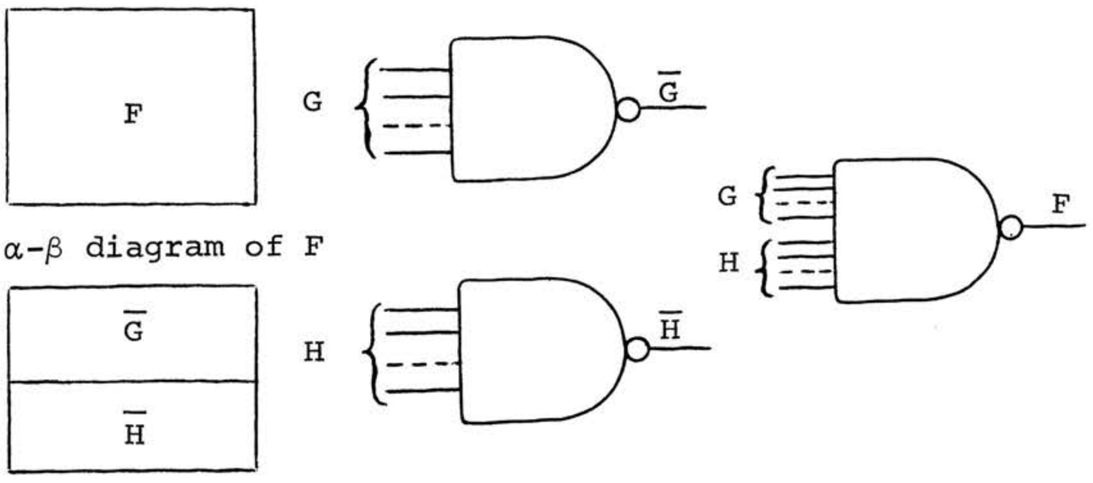


Figure 18. Decomposition of a diagram

In decomposing the α-β diagram one should usually attempt to form smaller α-β diagrams which rank higher on the priority list (see page 23) than the original diagram.

2. Extension to NOR Synthesis

The NOR function is the dual of the NAND. To synthesize a given function using NOR gates, obtain the α-β diagram as in synthesis with NANDs. Form the dual α-β diagram before any simplification is employed. As previously described, the dual α-β diagram is formed by simply interchanging the α-set and β-set. The dual α-β diagram is then simplified as described for NAND synthesis. The algorithm can then be applied directly to the dual α-β diagram, substituting NOR gates for NANDs in the final network realization. This procedure is simply that of taking the dual of the given function twice, once forming the dual α-β diagram and once by replacing NAND gates with NOR gates.

3. Worked Examples

In order to illustrate the diagrammatic procedure presented in this paper, three examples will be worked in detail.

EXAMPLE PROBLEM 1. Design a NAND circuit which realizes the switching function, x, shown mapped below.

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	1	1	0	1
10	0	0	0	1

There is only one minimal sum of products expression and one minimal product of sums expression. Therefore, the α-set is easily determined as

$$\{AB\overline{C},\ B\overline{C}D,\ AC\overline{D}\}$$

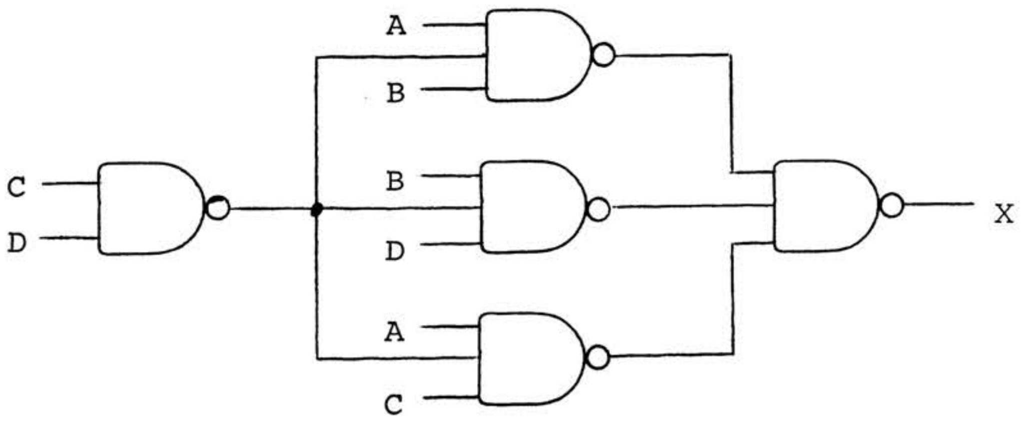
and the β-set is

$$\{\overline{C}\overline{D},\ BC,\ AD\}$$

The α-β diagram is then constructed and is shown below. The row and column labels are retained for clarity.

	$\overline{C}\overline{D}$	BC	AD
$AB\overline{C}$	$\overline{C}$	B	A
$B\overline{C}D$	$\overline{C}$	B	D
$AC\overline{D}$	$\overline{D}$	C	A

Since each square contains only one literal no simplification is necessary. The algorithm is then applied to the α-β diagram. The conditions of algorithm step 7 are the first to be satisfied and the network is synthesized by step 8. The NAND realization is



This function has been synthesized by Ellis (7). In that work the solution was arrived at by a partly systematic and partly trial and error procedure. The synthesis by the algorithm of this paper is simple and straight-forward, following only well defined steps.

EXAMPLE PROBLEM 2. Design a NAND circuit which realizes the switching function, y, shown mapped below.

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	1	1
10	1	1	1	1

Solution: The α-set is determined from the logical multiplication of the two minimal sum of products expressions

$$y_1 = \overline{D} + \overline{A}B + A\overline{B} + AC$$

$$y_2 = \overline{D} + \overline{A}B + A\overline{B} + BC$$

After this multiplication and the allowed simplification

$$y_1 y_2 = \overline{D} + \overline{A}B + A\overline{B} + ABC$$

from which the α-set is

$$\left\{\overline{D},\ \overline{A}B,\ A\overline{B},\ ABC\right\}$$

There is one minimal product of sums expression from which the β-set is determined as

$$\left\{AB\overline{D},\ \overline{A}\overline{B}C\overline{D}\right\}$$

The α-β diagram is then constructed and is shown below.

	$AB\bar{D}$	$\bar{A}\bar{B}C\bar{D}$
$\bar{D}$	$\bar{D}$	$\bar{D}$
$A\bar{B}$	A	$\bar{B}$
$\bar{A}B$	B	$\bar{A}$
ABC	AB	C

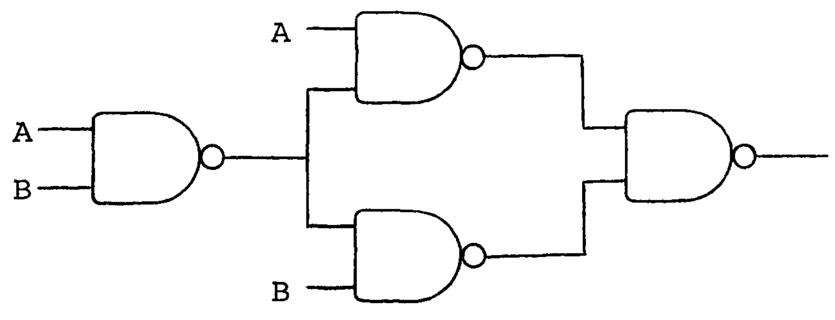
The removal of either A or B in the bottom row is arbitrary. For this example, B will be removed. The algorithm is then applied to the α-β diagram and the complemented row, $\bar{D}$, is removed by step 4. The resulting diagram is

A	$\bar{B}$
B	$\bar{A}$
A	C

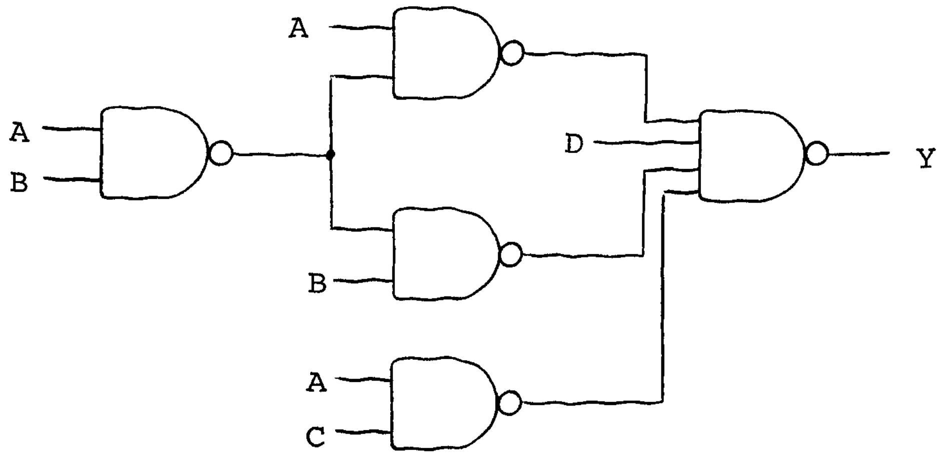
to which the algorithm is applied. The conditions of step 9 are the first to be satisfied and the uncomplemented row is removed by step 10. The remaining diagram

A	$\bar{B}$
B	$\bar{A}$

is synthesized by step 8. The result of thissynthesis is



With the addition of the gate for the uncomplemented row and the input to the output gate for the complemented row, the synthesis is complete and the NAND realization is



EXAMPLE PROBLEM 3. Design a NAND network which realizes the switching function, z, shown mapped below.

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	1	0	1
11	0	0	1	0
10	0	1	0	1

The α-set is easily determined from the two minimal sum of products expression as

$$\left\{\bar{A}\bar{B}\bar{C}\bar{D},\ \bar{A}\bar{C}D,\ \bar{A}C\bar{D},\ \bar{B}\bar{C}D,\ \bar{B}C\bar{D},\ ABCD\right\}$$

There is one product of sums expression from which the β-set is determined as

$$\left\{A\bar{C}\bar{D},\ B\bar{C}\bar{D},\ \bar{A}\bar{B}D,\ \bar{A}\bar{B}C,\ \bar{A}CD,\ \bar{B}CD\right\}$$

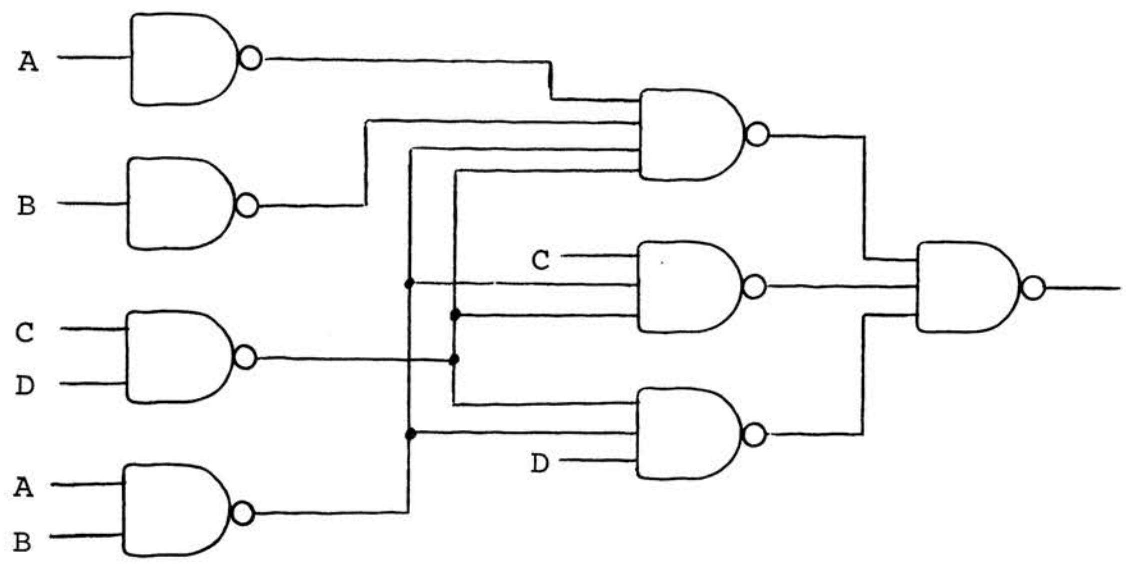
The α-β diagram before simplification is

	$A\bar{C}\bar{D}$	$B\bar{C}\bar{D}$	$\bar{A}\bar{B}D$	$\bar{A}\bar{B}C$	$\bar{A}CD$	$\bar{B}CD$
$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{C}\bar{D}$	$\bar{C}\bar{D}$	$\bar{A}\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}$	$\bar{B}$
$\bar{A}\bar{C}D$	$\bar{C}$	$\bar{C}$	$\bar{A}D$	$\bar{A}$	$\bar{A}D$	D
$\bar{A}C\bar{D}$	$\bar{D}$	$\bar{D}$	$\bar{A}$	$\bar{A}C$	$\bar{A}C$	C
$\bar{B}\bar{C}D$	$\bar{C}$	$\bar{C}$	$\bar{B}D$	$\bar{B}$	D	$\bar{B}D$
$\bar{B}C\bar{D}$	$\bar{D}$	$\bar{D}$	$\bar{B}$	$\bar{B}C$	C	$\bar{B}C$
$ABCD$	A	B	D	C	CD	CD

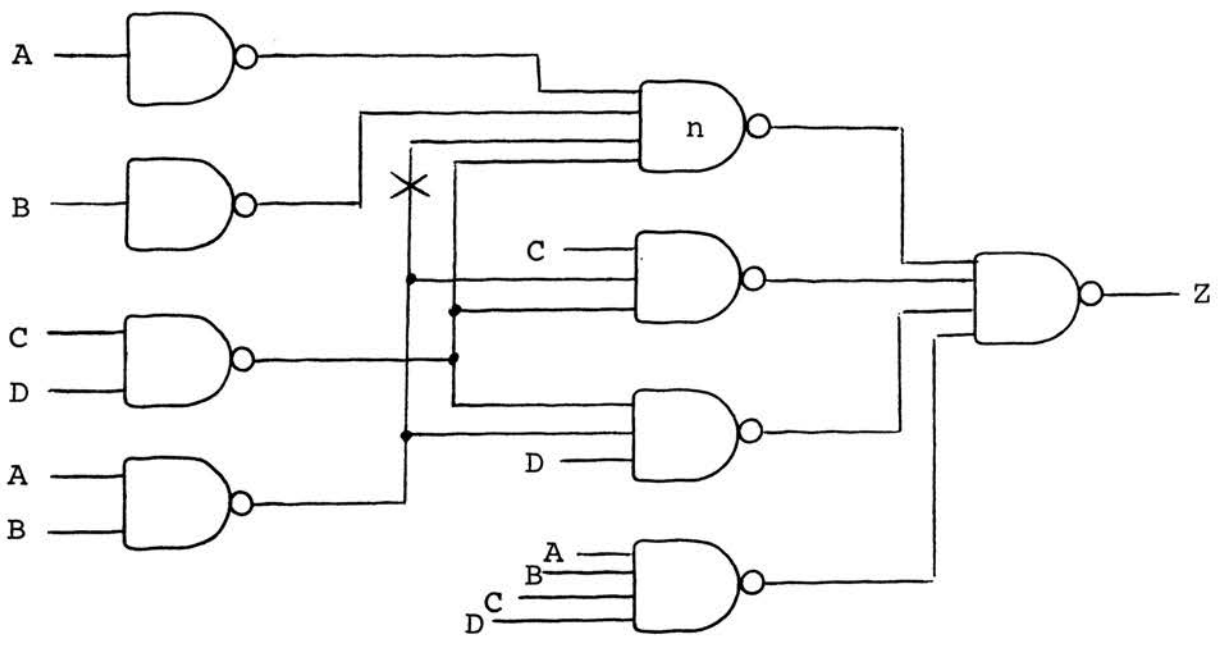
The redundant literals are removed according to the simplification priority. The diagram for z is then

$\bar{C}$	$\bar{C}$	$\bar{A}$	$\bar{A}$	$\bar{A}$	$\bar{B}$
$\bar{C}$	$\bar{C}$	$\bar{A}$	$\bar{A}$	D	D
$\bar{D}$	$\bar{D}$	$\bar{A}$	$\bar{A}$	C	C
$\bar{C}$	$\bar{C}$	$\bar{B}$	$\bar{B}$	D	D
$\bar{D}$	$\bar{D}$	$\bar{B}$	$\bar{B}$	C	C
A	B	D	C	D	D

The uncomplemented row is removed by step 10 and the resulting diagram is synthesized by step 12 as



With the addition of the gate for the removed uncomplemented row, the final network is



The input to gate n marked by an X is redundant and is removed by input reduction rule 1. This function has been synthesized on page 149 of Maley and Earle (4) using a map factoring technique. Considerably less effort was involved using the technique of this paper.

CHAPTER III

SUMMARY

The general approach to the synthesis of NAND logic networks presented in this paper differs considerably from the algebraic and map methods. It has been assumed here that complemented variables are not available as network inputs. Akers was the first to suggest synthesis of Boolean functions through the use of an α-β diagram. The author's development is a modification of Akers' work to produce minimal NAND-NOR networks. This diagrammatic approach yields minimal or near minimal networks without the complexities of trial and error procedures inherent in the algebraic and map methods. The synthesis algorithm of this paper consists of a set of well defined steps leading directly from an arbitrary Boolean expression to the final network. It is applicable to both completely and incompletely specified Boolean functions.

BIBLIOGRAPHY

1. SMITH, R. A., Minimal Three-Variable NOR and NAND Logic Circuits, IEEETEC, Vol. EC-14, No. 1, pp. 79-82, February, 1965.

2. DIETMEYER, D. L. and SCHNEIDER, P. R., A Computer Oriented Factoring Algorithm for NOR Logic Design, IEEETEC, Vol. EC-14, No. 6, pp. 868-874, December, 1965.

3. HELLERMAN, L., A Catalog of Three Variables OR-Invert and AND-Invert Logical Circuits, IEEETEC, Vol. EC-12, No. 3, pp. 198-223, June, 1963.

4. MALEY, G. A. and EARLE, J., The Logical Design of Transistor Digital Computers, Prentic Hall, Englewood Cliffs, Jew Jersey, 1963.

5. McCLUSKEY, E. J., Logical Design Theory of NOR Gate Network with No Complemented Inputs, Proc. Fourth Annual Symposium on Switching Circuit Theory and Logical Design, Chicago, Illinois, pp. 137-148, October, 1963.

6. GIMPEL, J. F., The Synthesis of TANT Networks, IEEE Conference Record on Switching Theory and Logical Design, Ann Arbor, Michigan, pp. 105-125, October, 1965.

7. ELLIS, D. T., A Synthesis of Combinational Logic with NAND or NOR Elements, IEEETEC, Vol. EC-14, No. 5, pp. 701-705, October, 1965.

8. AKERS, S. B., Jr., A Diagrammatic Approach to Multilevel Logic Synthesis, Proc. of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, pp. 165-173, October, 1964. IEEETEC, Vol. EC-14, No. 2, pp. 174-181, April, 1965.

9. AKERS, S. B., Jr., A Truth Table Method for the Synthesis of Combinational Logic, IEEETEC, Vol. EC-10, No. 4, pp. 604-615, December, 1961.

VITA

The author was born on October 21, 1940, in Berwyn, Illinois. He received his primary education in Berwyn, Illinois and graduated from J. Sterling Morton High School in Cicero, Illinois in 1958. He was awarded the Iowa State Club of Chicago Scholarship and obtained a Bachelor of Science degree from Iowa State University in Ames, Iowa in May, 1962.

He was enrolled in the Graduate School of the University of Missouri at Rolla in September, 1962 and held a graduate assistantship in Electrical Engineering until June, 1963. He joined the staff of the Electrical Engineering Department in September, 1963 as a half time instructor and since June, 1964 has held a full time appointment.

Mr. Nelson is a member of Phi Eta Sigma, Kappa Mu Epsilon, Tau Beta Pi, Eta Kappa Nu, and the Institute of Electrical and Electronics Engineers.