MULTILEVEL-CIRCUIT DESIGN USING LOGIC TREES

by 632

HARRY LEE PUETT

B. S., Kansas State University, 1969

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1970

Approved by:

L. A. Wirth

Major Professor

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

A number of different logic design aids have been devised and have been used in the past few years. The constant need for better and more economical logic circuits causes a never ending search for better design methods. This paper reports on the work of Green and Foulk (2) which deals with the uses of logic trees in multilevel-circuit design. These logic trees employ only 2-input 'AND' and 'OR' logic gates and are capable of mechanizing any desired Boolean function. The logic tree provides a standard adaptive-logic array for Boolean functions. Computer aided design procedures use the logic tree as a standard circuit, and multilevel representations of Boolean functions are derived using the logic tree (2). An original computer program which minimizes Boolean functions using the logic tree procedure is included in this report.

THE DERIVATION OF LOGIC TREES

This section presents a general derivation of the basic equations of the logic tree and the implementation of the basic equations using logic gates.

Let F_n represent a general Boolean function. F_n has n inde pendent variables X_i, where i = 1, 2, 3, . . ., n. F_n can be expressed in canonical form as a sum of products. Since F_n can be represented in this way, the general structure of F_n can be expressed as shown in Table 1. The X_i' means 'not X_i', '+' is the OR function, and '.' is the AND function. The K_i terms in Table 1 are either zero or one. For an example, consider the

term $X_2 \cdot X_1$ in line (c) of Table 1. If the term $X_2 \cdot X_1$ is not a term in the canonical sum-of-products, the value of K_3 is zero. Thus the K_i terms are coefficients in the minterm expansion of the Boolean function being represented by F_n. If the proper choice of the values of the K_i variables is made, F_n can represent any Boolean function which contains n of the X_i variables (2).

The key to the logic tree is shown in equation (1). F_j^* is the same general function as F_j, but F_j^* and F_j have a different set of K_i coefficients.

$$F_n = X_n' \cdot F_{n-1} + X_n \cdot F_{n-1}^* \tag{1}$$

The proof of equation (1) follows. Let line (b) in Table 1 be F_{n-1}. Then

$$F_{n-1} = F_1 = X_1' \cdot K_0 + X_1 \cdot K_1 \tag{2}$$

and

$$F_{n-1}^* = F_1^* = X_1' \cdot K_2 + X_1 \cdot K_3 \ . \tag{3}$$

Therefore

$$F_2 = X_2' \cdot (X_1' \cdot K_0 + X_1 \cdot K_1) + X_2 \cdot (X_1' \cdot K_2 + X_1 \cdot K_3). \tag{4}$$

If equation (4) is expanded,

$$F_2 = X_2' \cdot X_1' \cdot K_0 + X_2' \cdot X_1 \cdot K_1 + X_2 \cdot X_1' \cdot K_2 + X_2 \cdot X_1 \cdot K_3 \tag{5}$$

which agrees with line (c) in Table 1. Thus equation (1) is true. Figure 1 shows the implementation of equation (1) using

Table 1

The Canonical Sum-of-Products Expressions for Each F_n

F_n	Line
$F_0 = K_0$	(a)
$F_1 = X_1' \cdot K_0 + X_1 \cdot K_1$	(b)
$F_2 = X_2' \cdot X_1' \cdot K_0 + X_2' \cdot X_1 \cdot K_1 + X_2 \cdot X_1' \cdot K_2 + X_2 \cdot X_1 \cdot K_3$	(c)

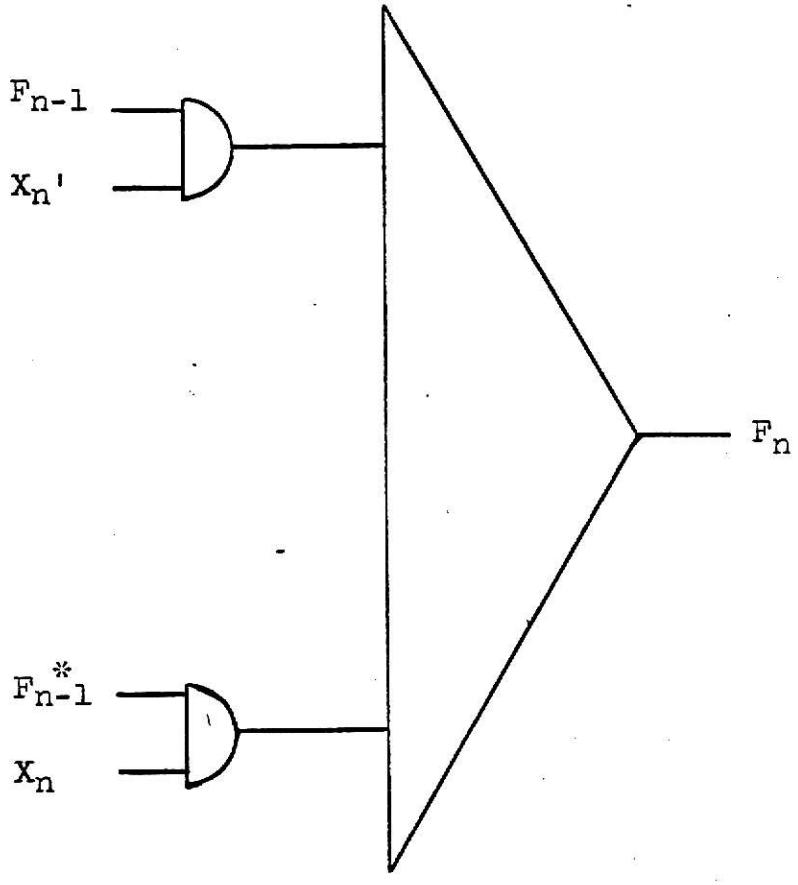


Fig. 1. The implementation of equation (1). (Ref. 2.)

logic gates. The recursion of the F_n's in equation (1) and the implementation shown in Fig. 1 are combined in order to build up a general logic tree for n variables. The logic tree for 4 variables is presented in Fig. 2. The implementation contains only 2-input AND and OR gates. The circuits within the dashed lines represent the logic trees for 1, 2, and 3 variables, respectively. If a Boolean function has 4 independent variables, the realization of the function is at the terminal marked output in Fig. 2. For functions with 1, 2, or 3 independent variables, the output is taken at the position marked F_1, F_2, or F_3, respectively (2).

The order of a Boolean function refers to the number of independent variables in the function. The order of the variables in the logic tree refers to the order in which the independent variables appear in the logic tree. For example in Fig. 2, the order of the function is four, and the order of the variables is X_1, X_2, X_3, and X_4.

The AND-OR tree shown in Fig. 2 is converted to a NAND element tree as shown in Fig. 3. The NAND element tree may be more economical to build than the AND-OR tree. The NAND tree has the same output and the same properties as the AND-OR tree. An explanation of the properties of the logic tree is presented in the following section.

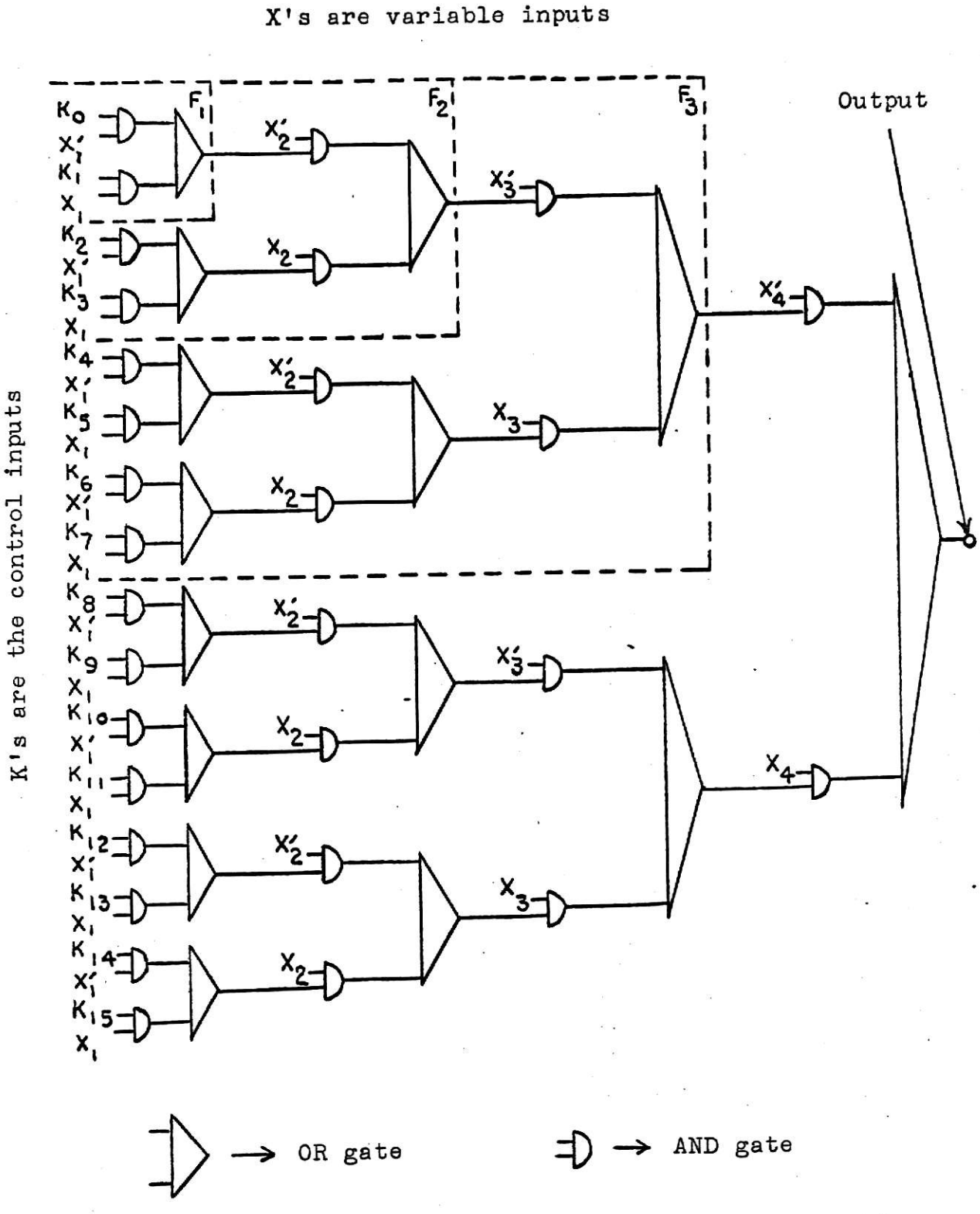
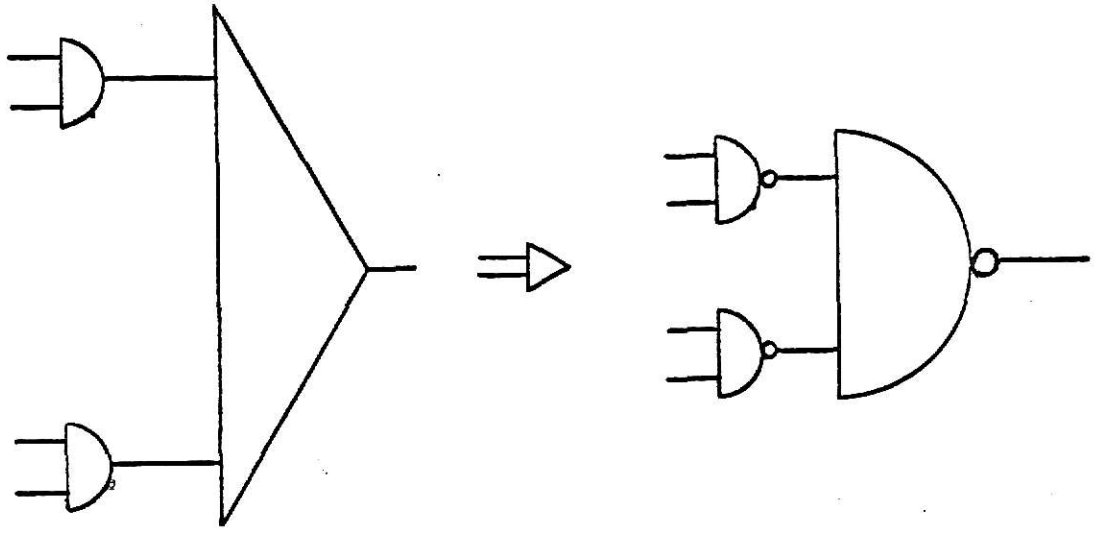
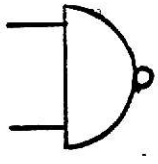


Fig. 2. The logic tree for four variables (2).





NAND gate

Fig. 3. Conversion of AND-OR tree to NAND element tree.

APPLICATIONS OF LOGIC TREES

The logic tree provides a method of minimizing Boolean functions as does the Karnaugh map method and the Quine-McCluskey method, but the logic tree provides design applications which other minimization methods do not provide.

Standard Adaptive Array

One application of the logic tree is as a standard adaptive-logic array. The standard array may represent any Boolean function by applying the K_i coefficients of the minterm expansion to the appropriate control inputs of the logic tree. If the control inputs are fixed, the logic tree mechanizes the appropriate function. For most Boolean functions, some of the gates are redundant; therefore the operation of the logic tree is maintained in spite of certain gate or component failures. Large networks may be made by combining smaller logic tree arrays. The size of the smaller array would be determined by the design technology. A manufacturer could mass produce a standard adaptive-logic array that is capable of representing any desired Boolean function. The mass production of the same circuit may have certain advantages. The logic array has become increasingly important with the development of the large-scale integration technology which affords special advantages for circuit arrays like high packing density, high reliability, high manufacturing yield, ease of error diagnosis, and flexibility in performance (2).

Standard Logic Circuit

Logic trees are useful in computer-aided design applications related to logic circuits. A computer-aided design application determines whether or not a certain logic circuit will work as planned without building an actual workin circuit. A computer program accepts the design specifications of a logic circuit and returns the value of the output of the circuit. Logic trees, as shown in Fig. 2, provide a standard means of displaying the circuit of a given logic function by relating the canonical form directly to the logic connections. A standard circuit is helpful in a computer-aided design procedure A computer programmer is able to write a program that analyzes a standard logic circuit, but a program which analyzes any given logic circuit is almost impossible to write (2).

Minimization of Boolean Functions

Another application of the logic tree is the minimization of Boolean functions. The logic tree method of minimization produces a multilevel realization. A multilevel expression is actually a factored form of a sum-of-products representation. As an example of the logic tree minimization procedure, a Boolean function called function F is minimized. F is a function of order four whose sum-of-products canonical form in decimal specification is

$$F = \sum(1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14) \quad (6)$$

Let A, B, C, and D be the independent variables of the Boolean function. The logic tree minimization of function F is shown in Fig. 4. Starting from the canonical form, the coefficients

of the minterms are entered at the control inputs of the logic tree appropriate to the chosen order of the independent variables. If the order of the variables is A, B, C, and D, the control inputs are as shown in Fig. 4. In the following discussion, let X_i represent any given independent variable. The first column of AND and OR gates in the logic tree are always redundant, since the outputs of the first column of OR gates are always either 0, 1, X_i, or X_i', as can be seen from the Boolean relationships

$$X_i' \cdot 0 = X_i \cdot 0 = 0 \tag{7}$$

$$X_i' + 1 = X_i + 1 = 1 \tag{8}$$

$$X_i' \cdot 1 = X_i' + 0 = X_i' \tag{9}$$

$$X_i \cdot 1 = X_i + 0 = X_i \tag{10}$$

$$X_i' \cdot X_i = 0 \tag{11}$$

$$X_i' + X_i = 1 \tag{12}$$

In Fig. 4, the redundant gates are shaded (2).

The simple relationships in equations (7) through (12) can eliminate some of the gates beyond the first column of OR gates, but other gates may possibly be eliminated by using the Boolean relationships.

$$X_i + X_i' \cdot X_j = X_i + X_j \tag{13}$$

$$X_i \cdot X_j + X_i' \cdot X_j = X_j \tag{14}$$

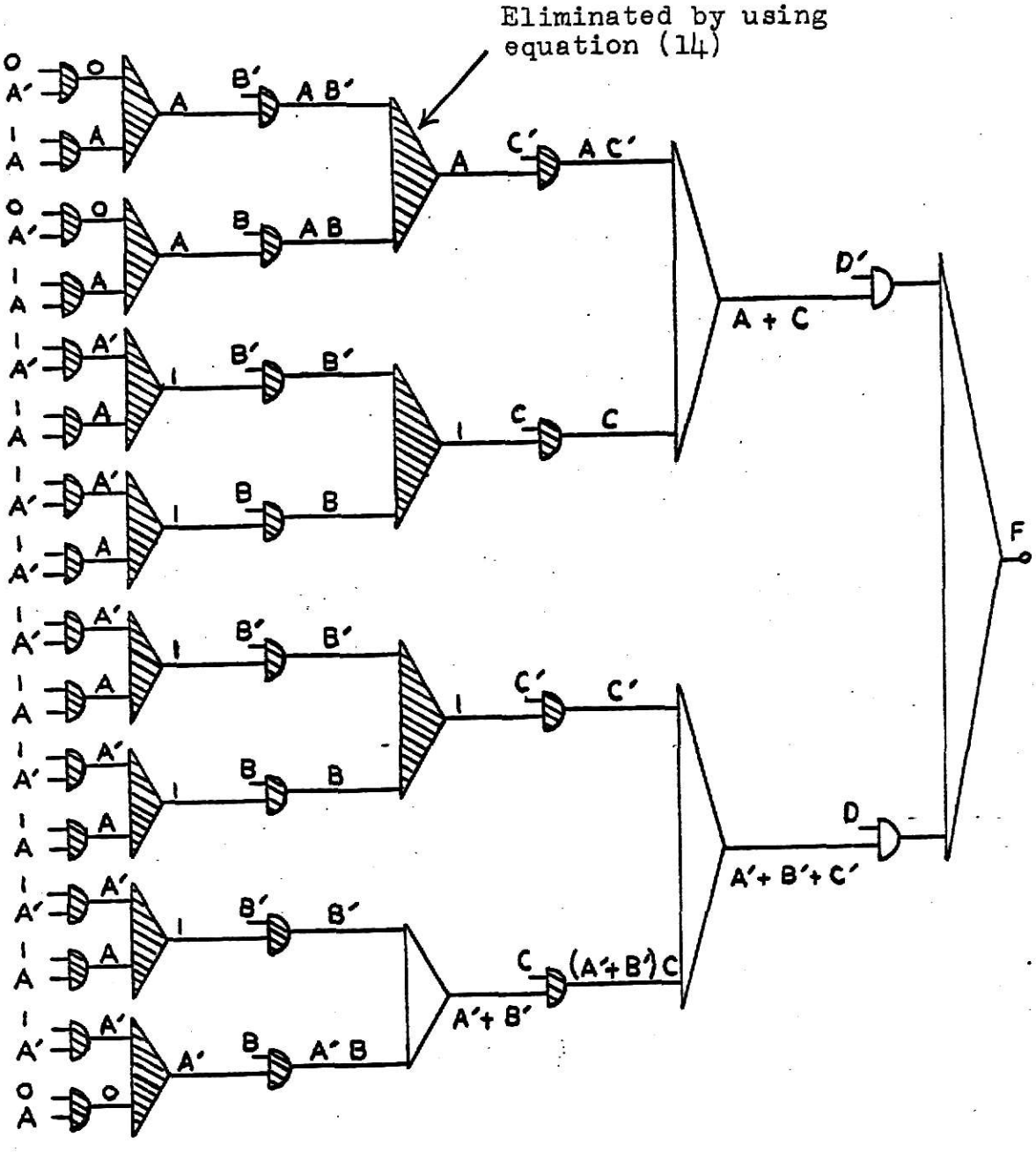
In Fig. 4, an arrow points to an OR gate which is eliminated by applying the Boolean relationship in equation (14). The multilevel solution is arrived at by removing the redundant gates from the logic tree. The multilevel solution of function F is shown in Fig. 5.

In some cases, the order of the independent variables may be changed which may produce a more economical solution. A reordering of the variable inputs is equivalent to factoring the variables out in a different order by using Boolean algebra. Thus one Boolean function may have several different multilevel solutions. A computer program which performs the logic tree minimization method is presented in the next section. This computer program allows the designer to investigate several multilevel solutions of a Boolean function (2).

TREE MINIMIZATION PROGRAM

The computer program presented in this report is an original program written by the author of this report. The name of the computer program is the Tree Minimization Program. The program is written in PL/I (Programming Language One). The Tree Minimization Program synthesizes any arbitrary switching function using a minimum number of 2-input AND and OR gates. A listing of the program is in Appendix 2 and a general flow chart of the program is in Appendix 1.

For the purpose of explanation, the program is divided into six sections. Each section is discussed in a general manner. The easiest way to follow the discussion is to relate



$$F = \Sigma(1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$$

$$F = D'(A + C) + D(C' + (A' + B'))$$

Fig. 4. The logic tree minimization of function F in equation (6).

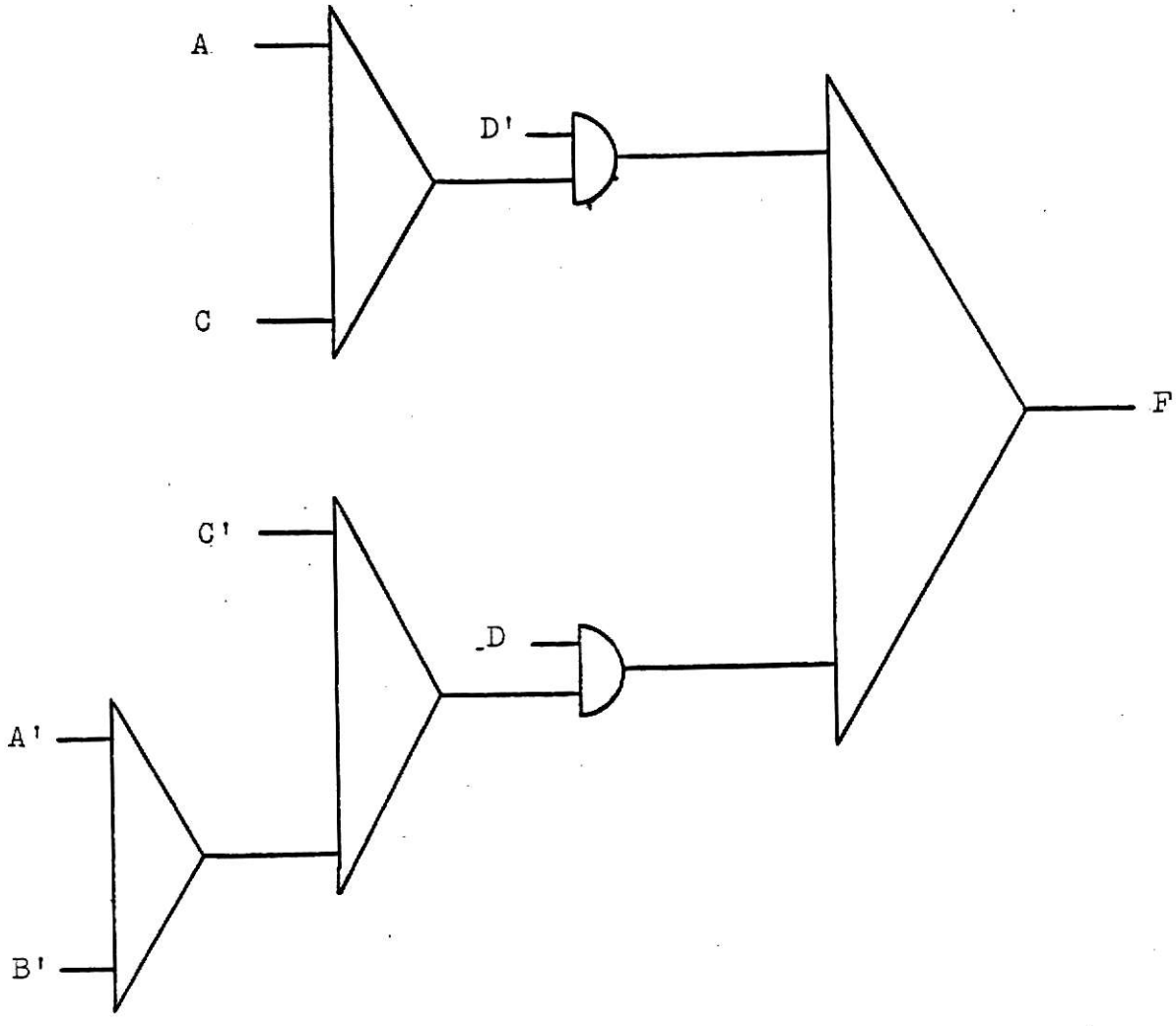


Fig. 5. The multilevel solution of function F.

the explanation of the program to the flow chart and then to relate the flow chart to the program listing.

In section 1, an array 'A' is initialized for later use in the calculation of the control inputs of the logic tree. Data that pertains to a given Boolean function is read from data cards. The contents of the data is the order of the Boolean function, the decimal specification of the canonical form of the function, and the numerical order of the independent variables in the logic tree. For an example of the data values, refer to the function F represented by the logic tree in Fig. 4. The order of the function F is four. The decimal specification of the canonical form is 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14. The order of the independent variables is A, B, C, and D; therefore the decimal order of the variables that is placed on the data card is 0, 1, 2, and 3. After the data is read, the control inputs of the logic tree are calculated and placed in the array STACK1. After the execution of section 1, section 2 is executed.

In section 2 of the Tree Minimization Program, the control inputs in STACK1 are tested; and the appropriate results are placed in the array STACK2. The elements in STACK1 are tested in pairs. The first two elements are tested, and the appropriate results are placed in the first element of STACK2. Then the second two elements in STACK1 are tested, and the appropriate results are placed in the second element of STACK2. This process continues until all of the control inputs are tested. The tests that are performed on the elements of STACK1 are

applications of the Boolean relationships in equations (7) through (12). The process of applying the Boolean relationships to the elements in STACK1 is equivalent to elimination of the first column of AND and OR gates. Thus STACK2 contains the outputs of the first column of OR gates. After all of the elements in STACK1 are tested, execution of the program goes to section 3.

Section 3 contains two loops which are named LOOP3 and LOOP2. LOOP2 is within LOOP3. In LOOP2, the elements in STACK2 are tested in pairs. The first two elements are tested, and the appropriate results are placed in the first element of STACK2. Then the second two elements in STACK2 are tested and the appropriate results are placed in the second element of STACK2. This process continues until all of the elements in STACK2 are tested. Once all the elements in STACK2 are tested, LOOP2 is executed again; and the new elements in STACK2 are tested. Thus the number of elements in STACK2 are decreased by a factor of two each time LOOP2 is executed. The process of executing LOOP2 continues until only one element is in the array STACK2. This final element in STACK2 is the multilevel solution. The results are placed back into STACK2 in order to save computer storage space. The tests that are performed on the elements of STACK2 are applications of the Boolean relationships in equations (7) through (12) and of the more complex Boolean relationships in equations (13) and (14). For an example of the process in LOOP2, refer to Fig. 6. When the multilevel solution is found, LOOP3 is terminated, and execution passes to section 4.

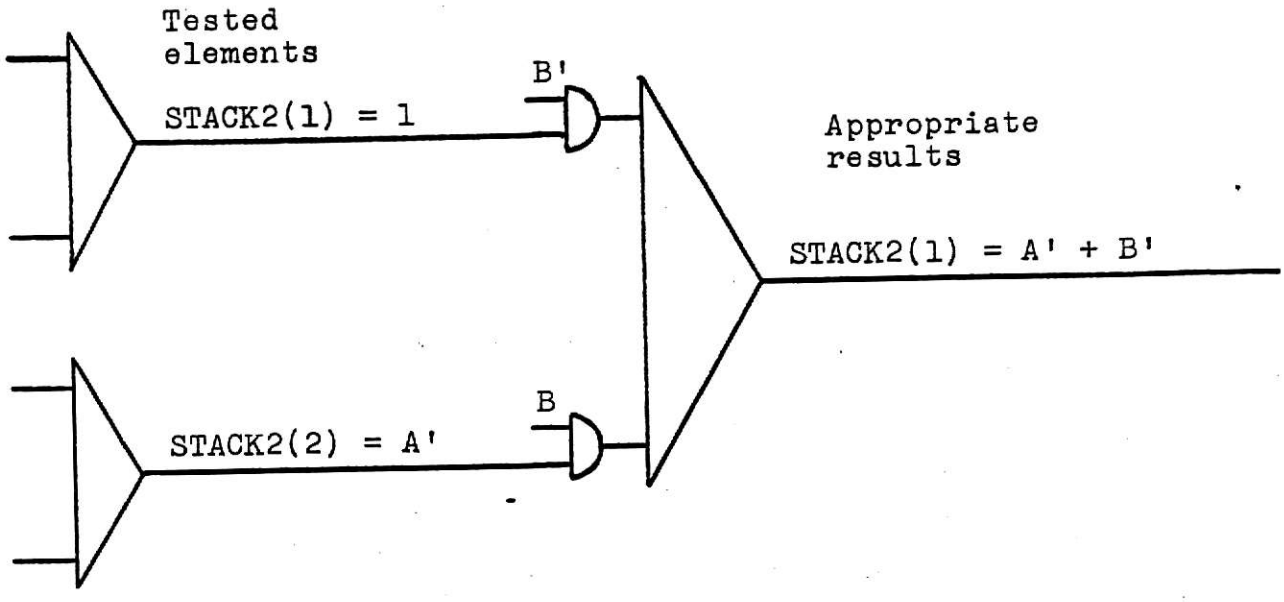


Fig. 6. An example of the process in LOOP2.

In section 4, the multilevel solution is printed out. In the computer print out, the variables which are followed by a zero represent primed variables; and the variables which are followed by a one represent unprimed terms.

Section 5 of the program contains a subroutine named TIMES. When TIMES is called by the main program, the quantity which has the variable name QUAN is multiplied term by term by the quantity which has the variable name MULP. The result of the multiplication is placed in the variable named E-STRING.

When all of the data cards have been read, section 6 of the program is executed. The statements in section 6 terminate the program.

The Tree Minimization Program can be used effectively in a logic design procedure. With the aid of this program, a logic designer can investigate several different multilevel circuit configurations of the same Boolean function. Since many different logic design procedures are available for use by the logic designer, a comparison of the logic tree minimization method to other methods of Boolean function minimization is presented in the next section.

COMPARISON OF MINIMIZATION METHODS

The general problem of designing economical switching networks has been solved for some important subcases, but essentially the problem is still an open one. Many different approaches to the minimization problem have been developed. In this section, the tree-minimization algorithm is compared to four other minimization procedures.

Two methods which are used to obtain minimal sum-of-products representations of Boolean functions are the Quine-McCluskey procedure and the Karnaugh map method. The Karnaugh map proves to be adequate if the number of independent variables is five or less. If the order of the function is six or greater, the analysis of the Karnaugh maps is unwieldly even if a computer-driven analysis is used. On the other hand, the Quine-McCluskey minimization procedure can be used effectively in a computer-driven design procedure. The major difference between the Quine-McCluskey minimization procedure and the logic tree minimization procedure is that the Quine solution is a minimal sum-of-products representation and that the logic tree solution is a multilevel representation. A multilevel solution is usually more economical to implement than a minimal sum-of-products solution for Boolean functions of order four or greater. More economical means that the circuit can be implemented with fewer logic gates. Five functions of order four and two functions of order five were minimized by using the Quine procedure and by using the logic tree procedure. The minimal representations of the seven Boolean functions are shown in Table 2. Table 2 shows that the minimal switching functions produced by the logic tree algorithm required fewer logic gates than the minimal switching functions produced by the Quine minimization procedure. Consider the first Boolean function in Table 2 whose canonical form is

$$F_1 = \Sigma(0, 2, 3, 4, 6, 10, 11, 12, 13, 15) \qquad (15)$$

In Table 2, the minimal sum-of-products solution of F_1 is

$$ABD + B'CD + A'D' + BC' \tag{16}$$

In this form, the solution requires six 2-input AND gates and three 2-input OR gates, or a total of nine logic gates. Of course, the solution can be factored. One of the factored forms of equation (16) that was found is

$$(AB + B'C)D + A'D' + BC' \tag{17}$$

This form of the solution requires five 2-input AND gates and three 2-input OR gates, or a total of eight logic gates. Equation (17) is an economical expression for the switching function F_1, but the factoring of the solution was not a straightforward process. A trial-and-error method must be used to factor equation (16). The factorization of a minimal sum-of-products representation becomes more difficult as the order of the Boolean function increases. The factorization of solutions of functions which are of order eight or greater is almost an impossibility. On the other hand, the logic tree multilevel solution of F_1 shown in Table 2 is

$$A'D' + BC' + (AD + A'B')C \tag{18}$$

Equation (18) does not require any factoring. The implementation of equation (18) requires five 2-input AND gates and three 2-input OR gates. The implementation of the logic tree solution of function F_1 and the implementation of the factored Quine solution of function F_1 both required eight logic gates. Thus

Table 2

Minimal Switching Functions Produced by the Logic Tree Minimization Procedure and by the Quine Minimization Procedure

Canonical Form	Quine Solution	Logic Tree Solution
(0,2,3,4,6,10,11,12,13,15)	ABD + B'CD + A'D + BC'	A'D' + BC' + (AD + A'B')C
(1,3,4,5,6,7,8,9,10,11,12,13,14)	AD' + A'C + B'D + C'D	(A + B)D' + (A' + B' + C')D
(0,1,2,3,8,10,11,17,19,21, 23,29,30,31)	ACE + AC'D' + A'C'E' + BCDE + BC'E'	(A' + B + D')C'E' + [AD' + (A + BD)C]E
(7,11,15,18,23,24,26,27,31)	ABC + ABD + A'BC'E + A'C'DE	(C + D)AB + [A'B + (A' + B)D]C'E
(2,3,5,7,9,11,14,15)	BC'D' + ACD' + AC'D + BCD	(BC' + AC)D' + (AC' + BC)D
(2,3,6,7,8,9,10,11,12,14)	BD' + C'D + A'D	(BD' + (C' + A')D
(0,2,4,8,12,13,15)	A'C'D' + ACD + A'B'	(B' + C'D')A' + (A + B')CD

a designer can investigate several economical multilevel representations of a Boolean function just by examining the output of the Tree Minimization Program.

Other minimization techniques which produce multilevel solutions have been developed in past years. One approach to the problem of designing multilevel realizations is the decomposition chart of Ashenhurst (1). Ashenhurst introduced the decomposition procedure in 1959. One disadvantage of Ashenhurst's procedure is that the technique is applicable only to a restricted class of functions. Another disadvantage is that the procedure requires the testing of $2^n - n - 2$ Karnaugh maps, where n is the number of independent input variables of a Boolean function. A recent modification of Ashenhurst's decomposition chart technique was introduced by Shen and McKellar (4). This modified procedure requires the testing of only $(n - 1)n/2$ Karnaugh maps, but the procedure is applicable only to a restricted class of functions. The logic tree minimization procedure is applicable to all Boolean functions.

A minimization method that was presented by Mukhopadhyay and Schmitz (3) is a technique which minimizes EXCLUSIVE OR and LOGICAL EQUIVALENCE switching functions. Mukhopadhyay and Schmitz have written a program which performs this minimization procedure. The program accepts the decimal specification of a switching and transforms the function to an EXCLUSIVE OR function or a LOGICAL EQUIVALENCE function. After the transformation, the program minimizes the transformed function and returns the solution as an EXCLUSIVE OR function or as a LOGICAL

EQUIVALENCE function. These solutions are economical switching functions, but a minimal cascade realization is not guaranteed. Algorithms which will guarantee minimal cascade realizations are currently under study by Mukhopadhyay and Schmitz.

The logic tree minimization procedure has one disadvantage that the minimization procedures mentioned above do not have. This disadvantage is that the tree minimization method can be used to minimize only functions of single output networks. Other minimization methods can be applied to multiple output networks. If only functions of single output networks are being considered in a logic network design, the Tree Minimization Program can be used efficiently to produce economical multilevel switching circuits without incurring any of the disadvantages that other minimization methods have.

SUMMARY

The logic tree has three major applications. The logic tree can be used as a standard adaptive-logic array for Boolean functions, as a standard circuit in computer-aided design applications, or as a minimization algorithm for Boolean functions. The latter of these applications was given the most attention in this report. A computer program that applies the minimization algorithm is presented in Appendix 2. Efficient multilevel logic circuits are obtained from the logic tree minimization procedure by using only the Boolean relationships in equations (7) through (14). Other minimization methods were discussed. A comparison of the different minimization methods showed that

the logic tree minimization procedure provided an economical single output switching function without having any of the disadvantages that the other minimization methods had. It is concluded that the logic tree is a useful aid in logic design procedures.

ACKNOWLEDGMENTS

I would like to thank Professor Leo A. Wirtz for his counsel and encouragement in the preparation of this report, and I am grateful for the invaluable assistance of Dr. Paul Fisher.

REFERENCES

1. R. L. Ashenhurst, "The Decompositions of Switching Functions," Proc. Internatl. Symp. on the Theory of Switching, Vol. 29 in Annals of Computation Lab. Cambridge, Mass.: Harvard University, 1959, pp. 74-116.

2. D. H. Green and P. W. Foulk, "Adaptive-Logic Trees for Use in Multilevel-Circuit Design," Electronics Letters, March 6, 1969, Vol. 5, No. 5, pp. 1-2.

3. Amar Mukhopadhyay and Greg Schmitz, "Minimization of EXCLUSIVE OR and LOGICAL EQUIVALENCE Switching Circuits," IEEE Transactions on Computers, February, 1970, Vol. C-19, No. 2, pp. 132-140.

4. V. Yun-Shen Shen and Archie C. McKellar, "An Algorithm for the Disjunctive Decomposition of Switching Functions," IEEE Transactions on Computers, March, 1970, Vol. C-19, No. 3, pp. 239-248.

APPENDICES

APPENDIX 1

A GENERAL FLOW CHART OF THE TREE MINIMIZATION PROGRAM

Section 1

```
Initialize array A.
Read Data (N, FNUM, ORDER).
N = order of function
FNUM = decimal specification of the canonical
       form of the Boolean function
ORDER = order of independent variables
The values of the control inputs are determined
and placed in the array STACK1.
```

Section 2

```
   J = 0
LOOP1: Do I = 1 TO 2**N
       J = J + 1
       Test STACK1(I) and STACK1(I + 1) and place
       the appropriate results in STACK2(J).
END of LOOP1
```

Section 3

```
   N1 = N
LOOP3: If N1 = 1 then go to Section 4
       N1 = N1 - 1
       Initialize variables
       J = 0
       Start LOOP 2
      Do I = 1 TO (2**N1) BY 2
         J = J + 1
         Test STACK2(I) and STACK2(I+1) using the
         Boolean relationships in equations (7)
         through (14) and place the appropriate
         results in STACK2(J).
         END of LOOP2
END of LOOP3
```

Section 4

Print out multilevel solution which is in STACK2(1).
Read Data and go to Section 1. If the end of data file is reached, go to Section 6.

Section 5

Subroutine TIMES
    E-STRING = (QUAN)MULP
    The multiplication is term by term.
RETURN

Section 6

Print out the statement Normal Termination.
END of the Tree Minimization Program

```
T_M_P: PROCEDURE OPTIONS (MAIN);

/*..............TREE MINIMIZATION PROGRAM..............*/

ON ENDFILE(SYSIN) GO TO TERM;
/* DECLARATION OF VARIABLES IS FOR FUNCTIONS OF ORDER 5 OR LESS. */
DCL (I,N,J,LE,LOW,LP,IS,LS,IC,SUB1,JF,N1,FUN,FUN1,TABLE,
CHAR1,CHAR2,CHARS,CHARL,CHARL1,REPLACE,P1,P,PS,PLUS,PSL,PA,PSA1,PSA2,
PAP,LENA,LENB,ST,IM,LENF,LENE) FIXED, LETTER CHAR(1), (LONG,LONGE,
SHORT,ENDS,ENDL,STACK2_I_1,STACK2_I,STRINGA,FACTOR,PARTB)
CHAR(100) VARYING, (TWO(5), A(32,5), FNUM(32)) FIXED,
(ORDER(5),ORDER1(5),STACK(32)) FIXED (1,0), (LETR,
 LFTOA,LET1A,LETL,LETS,PARTA,PARTA2,MULT) CHAR(2), (LETOP,LET1P)
CHAR(3), STACK1(32) CHAR(1), STACK2(16) CHAR(100) VARYING,
ALPHA(5) CHAR(1) INITIAL('A','B','C','D','E');

/*.....................SECTION 1 ......................*/

      /* INITIALIZE ARRAY A */
      DO J = 1 TO 5;
          IS = 0;
          LE = 0;
          DO WHILE (LE < 32);
              LOW = 1+IS;
              LP  = 2**(J-1)+IS;
              DO I = LOW TO LP;
                  A(I,J) = 0;
              END;
              LS = 2**(J-1)+1+IS;
              LE = 2**J+IS;
              IS = LE;
              DO I = LS TO LE;
                  A(I,J) =1;
              END;
          END;
      END;
      GET LIST(FUN);
      FUN1 = FUN;

      /* READ DATA */
      /* N = ORDER OF FUNCTION */
BEGIN: GET LIST(N);
      GET LIST(JF);
      PUT PAGE;
      PUT  LIST('NEW FUNCTION');
      PUT SKIP(2);
      PUT EDIT ('FUNCTION HAS ORDER',N) (A,F(2));
      PUT SKIP(2);
      PUT LIST ('CANONICAL FORM OF FUNCTION IS');
      PUT SKIP;
      /* FNUM = DECIMAL SPECIFICATION OF THE CANONICAL
         FORM OF THE BOOLEAN FUNCTION */
      DO I = 1 TO JF;
```

```
            GET LIST( FNUM(I) );
            PUT EDIT (FNUM(I),',') (F(2),A);
        END;
        PUT SKIP(2);
        FNUM(JF+1) =0;
        J = 1;
        DO I = 1 TO 2**N;
            IF FNUM(J) = I-1 THEN
                DO;
                    STACK(I) = 1;
                    J = J + 1;
                END;
                ELSE
                    STACK(I) = 0;
        END;
CVER:   N1 = N;
        PUT SKIP(2);
        PUT LIST('ORDER OF VARIABLES IN TREE');
        PUT SKIP;
        PUT LIST('AND OPPOSITE ORDER OF FACTORING THE SOLUTION');
        PUT SKIP;

        /* ORDER = ORDER OF INDEPENDENT VARIABLES */
        DO I = 1 TO N;
            GET LIST( ORDER(I) );
            ORDER1(I) = ORDER(I) +1;
            PUT EDIT (ALPHA(ORDER1(I)),',') (A,A);
            TWO(I) = 2**ORDER(I);
        END;
        PUT SKIP;

        /* THE VALUES OF THE CONTROL INPUTS ARE DETERMINED
           AND PLACED IN THE ARRAY STACK1 */
        DO I = 1 TO 2**N;
            TABLE = 0;
            DO J = 1 TO N;
                TABLE = TABLE + A(I,J) * TWO(J);
            END;
            J = TABLE + 1;
            IF STACK(J) = 1
                THEN STACK1(I) = '1';
                ELSE STACK1(I) = '0';
        END;


/*.....................SECTION 2 ......................*/


        /* STACK1(I) AND STACK1(I+1) ARE TESTED, AND THE
           APPROPRIATE RESULTS ARE PLACED IN STACK2(J). */
        J = 0;
        DO I = 1 TO 2**N BY 2;
            J = J+1;
            IF STACK1(I) = STACK1(I+1) THEN
                DO;
                    STACK2(J) = STACK1(I);
                    GO TO LOOP1;
```

```
            END;
          IF STACK1(I) = '1'
            THEN STACK2(J) = ALPHA(ORDER1(1))||'0';
            ELSE STACK2(J) = ALPHA(ORDER1(1))||'1';
LOOP1: END;

/*.....................SECTION 3.....................*/

      N1 = N;
      IC = 1;

      /* START OF LOOP3 */
LOOP3: IF N1 = 1 THEN
         GO TO OUTPUT;
      N1 = N1-1;
      IC = IC + 1;

      /* INITIALIZE VARIABLES */
      LETTER = ALPHA(ORDER1(IC));
      LET0P = '+'||LETTER||'0';
      LET1P = '+'||LETTER||'1';
      LET0A = LETTER||'0';
      LET1A = LETTER||'1';
      J = 0;

      /* START OF LOOP2 */
      DO I = 1 TO (2**N1) BY 2;
         J = J+1;

      /* STACK2(I) AND STACK2(I+1) ARE TESTED USING THE
         BOOLEAN RELATIONSHIPS IN EQUATIONS 7 THRU 12,
         AND THE APPROPRIATE RESULTS ARE PLACED IN STACK2(J). */
          STACK2_I_1 = STACK2(I+1);
          STACK2_I = STACK2(I);
          IF STACK2_I  = STACK2_I_1  THEN
             DO;
                STACK2(J) = STACK2_I ;
                GO TC LOOP2;
             END;
          IF STACK2_I  ¬= '0' & STACK2_I  ¬= '1' &
             STACK2_I_1  ¬= '0' & STACK2_I_1  ¬= '1'
             THEN GO TO CHECK;
          IF STACK2_I = '0' & STACK2_I_1 = '1'
             THEN
             DO;
                STACK2(J) = LET1A;
                GO TC LOOP2;
             END;
          IF STACK2_I = '1' & STACK2_I_1 ='0'
             THEN
             DO;
                STACK2(J) = LET0A;
                GO TC LOOP2;
             END;
          IF STACK2_I  = '1' THEN
```

```
            DO;
                STACK2(J) = STACK2_I_1||LETOP;
                GO TO LOOP2;
            END;
        IF STACK2_I_1  = '1' THEN
            DO;
                STACK2(J) = STACK2_I||LET1P;
                GO TO LOOP2;
            END;
        IF STACK2_I  = '0' THEN
            DO;
                MULT = LET1A;
                CALL TIMES(STACK2_I_1,MULT,STACK2(J));
                GO TO LOOP2;
            END;
        IF STACK2_I_1  = '0' THEN
            DO;
                MULT = LET0A;
                CALL TIMES(STACK2_I,MULT,STACK2(J));
                GO TO LOOP2;
            END;
        PUT LIST ('SOMETHING WRONG');
        GO TO TERM;
CHECK:  CHAR1 = LENGTH(STACK2(I));
        CHAR2 = LENGTH(STACK2(I+1));
        IF CHAR1 <= CHAR2 THEN
            DO;
                LONG = STACK2(I+1);
                SHORT = STACK2(I);
                CHARL = CHAR2;
                CHARS = CHAR1;
                LETL = LET1A;
                LETS = LET0A;
            END;
            ELSE
            DO;
                LONG = STACK2(I);
                SHORT = STACK2(I+1);
                CHARL = CHAR1;
                CHARS = CHAR2;
                LETL = LET0A;
                LETS = LET1A;
            END;
        REPLACE =0;
        LONG = '+'||LONG||'+';
        FACTOR = ' ' ;
        ENDS = ' ';
        P1 = 1;

    /* STACK2(I) AND STACK2(I+1) ARE TESTED USING THE
       BOOLEAN RELATIONSHIPS IN EQUATIONS 12 AND 13,
       AND THE APPROPRIATE RESULTS ARE PLACED IN STACK2(J). */
P_LOOP1: PS = INDEX(SUBSTR(SHORT,P1),'+');
        IF PS = 0 THEN
            STRINGA = SUBSTR(SHORT,P1);
            ELSE STRINGA = SUBSTR(SHORT,P1,PS-1);
        LENA = LENGTH(STRINGA);
```

```
          P1 = P1+PS;
PINDEX:   P = INDEX(LONG,STRINGA);
          IF P = 0 & CHARS = 2 THEN
               DO;
                    CHARL = LENGTH(LONG)-2;
                    LONG = SUBSTR(LONG,2,CHARL);
                    GO TO READY2;
               END;
          IF P = 0 THEN
               GO TO REPEAT;
          IF SUBSTR(LONG,P+LENA,1) = '+' &
             SUBSTR(LONG,P-1,1) = '+' THEN
               DO;
                    FACTOR = FACTOR||STRINGA||'+';
                    CHARL = LENGTH(LONG);
                    IF LENA = CHARL-2 THEN
                         DO;
                              LONG = '++';
                              GO TO P1TEST;
                         END;
                    LONG = SUBSTR(LONG,1,P-1)||SUBSTR(LONG,P+LENA+1);
                    IF LENA = CHARS THEN
                         DO;
                              ENDS = ' ';
                              CHARL = LENGTH(LONG)-2;
                              LONG = SUBSTR(LONG,2,CHARL);
                              GO TO READY3;
                         END;
                    P1TEST:  IF P1 = LENA+2 THEN
                         DO;
                              SHORT = SUBSTR(SHORT,P1);
                              P1 = 1;
                              GO TO PSCHECK;
                         END;
                    IF PS = 0 THEN
                         DO;
                              SHORT = SUBSTR(SHORT,1,CHARS-LENA-1);
                              GO TO PSCHECK;
                         END;
                    SHORT = SUBSTR(SHORT,1,P1-LENA-2)||SUBSTR(SHORT,P1);
                    P1 = P1-PS;
                    GO TO PSCHECK;
               END;
          PLUS = 1;
BACK:     IF SUBSTR(LONG,P-PLUS,1) ^= '+' THEN
               DO;
                    PLUS = PLUS+2;
                    GO TO BACK;
               END;
          PSL = INDEX(SUBSTR(LONG,P),'+');
          FACTOR = FACTOR||SUBSTR(LONG,P-PLUS+1,PLUS+PSL-2)||'+';
          CHARL = LENGTH(LONG);
          IF P-PLUS = 1 & P+PSL = CHARL+1 THEN
               DO;
                    LONG = '++';
                    GO TO PSCHECK;
               END;
```

```
          IF P-PLUS = 1 THEN
              LONG = '+'||SUBSTR(LONG,P+PSL);
              ELSE
              LONG = SUBSTR(LONG,1,P-PLUS-1)||SUBSTR(LONG,P+PSL-1);
          GO TO PINDEX;
 REPEAT:  ST = 1;
          PARTA = SUBSTR(STRINGA,1,2);
 REPEAT2: PA = INDEX(SUBSTR(LONG,ST),PARTA);
          IF PA = 0 THEN
              GO TO PSCHECK;
          PA = ST+PA-1;
          ST = PA+1;
          PSA1 =1;
   BACK2: IF SUBSTR(LONG,PA-PSA1,1) ¬= '+' THEN
              DO;
                   PSA1 = PSA1+2;
                   GO TO BACK2;
              END;
          PSA2 = INDEX(SUBSTR(LONG,PA),'+');
          PARTB = SUBSTR(LONG,PA-PSA1+1,PSA1+PSA2-2);
          LENB = LENGTH(PARTB);
          IF LENA = LENB THEN
              GO TO REPEAT2;
          IM = 3;
   BACK3: PARTA2 = SUBSTR(STRINGA,IM,2);
          PAP = INDEX(PARTB,PARTA2);
          IF PAP = 0 THEN
              GO TO REPEAT2;
          IM = IM+2;
          IF IM < LENA THEN
              GO TO BACK3;
          FACTOR = FACTOR||PARTB||'+';
          LONG = SUBSTR(LONG,1,PA-PSA1)||SUBSTR(LONG,PA+PSA2);
          ST = 1;
          GO TO REPEAT2;
 PSCHECK: CHARS = LENGTH(SHORT);
          CHARL = LENGTH(LONG);
          IF CHARL = 2 THEN
              DO;
                   CALL TIMES(SHORT,LETS,ENDS);
                   STACK2(J) = SUBSTR(FACTOR,2)||ENDS;
                   GO TO LOOP2;
              END;
          IF PS ¬= 0 THEN
              GO TO P_LOOP1;
          IF REPLACE = 1 THEN
              GO TO READY1;
          REPLACE = 1;
          CHARL1 = LENGTH(LONG)-2;
          LONGE = SHORT;
          SHORT = SUBSTR(LONG,2,CHARL1);
          LONG = '+'||LONGE||'+';
          CHARL = CHARS+2;
          CHARS = CHARL1;
          LETR = LETL;
          LETL = LETS;
          LETS = LETR;
```

```
          P1 = 1;
          GO TO P_LOOP1;
 READY1:  CHARL = LENGTH(LONG)-2;
          LONGE = SHORT;
          SHORT = SUBSTR(LONG,2,CHARL);
          LONG = LONGE;
          LETR = LETL;
          LETL = LETS;
          LETS = LETR;
 READY2:  CALL TIMES(SHORT,LETS,ENDS);
 READY3:  CALL TIMES(LONG,LETL,ENDL);
          LENF = LENGTH(FACTOR);
          LENE = LENGTH(ENDS);
          IF LENF = 1 & LENE = 1 THEN
              DO;
                  STACK2(J) = ENDL;
                  GO TO LOOP2;
              END;
          IF LENE = 1 THEN
              DO;
                  STACK2(J) = SUBSTR(FACTOR,2)||ENDL;
                  GO TO LOOP2;
              END;
          IF LENF = 1 THEN
              DO;
                  STACK2(J) = ENDS||'+'||ENDL;
                  GO TO LOOP2;
              END;
          STACK2(J) = SUBSTR(FACTOR,2)||ENDS||'+'||ENDL;

      /* END OF LOOP2 */
LOOP2: END;

      /* END OF LOOP3 */
      GO TO LOOP3;


/*.....................SECTION 4 ......................*/


OUTPUT: PUT LIST ('SOLUTION') SKIP;

      /* PRINT OUT THE MULTILEVEL SOLUTION IN STACK2(1). */
      PUT LIST (STACK2(1)) SKIP;
      PUT SKIP;
      PUT LIST ('THE ZEROS REPRESENT PRIMED TERMS AND') SKIP;
      PUT LIST ('THE ONES REPRESENT UNPRIMED TERMS') SKIP;
      PUT SKIP(2);

      /* READ DATA AND GO TO SECTION 1. IF THE END OF THE
         DATA FILE IS REACHED, GO TO TERM IN SECTION 6. */
      GET LIST (FUN1);
      IF FUN1 = FUN THEN
          GO TO OVER;
      FUN = FUN1;
      GO TO BEGIN;
```

```
/*.....................SECTION 5 ......................*/


      /* SUBROUTINE TIMES */
TIMES: PROCEDURE(QUAN,MULP,E_STRING);
DCL (QUAN,E_STRING,STRINGP) CHAR(100) VARYING, MULP CHAR(2),
(Q,Q1,CHARQ) FIXED;
      Q = INDEX(QUAN,'+');
      IF Q = 0 THEN
          DO;
              E_STRING = QUAN||MULP;
              RETURN;
          END;
      Q1 = 1;
      STRINGP = SUBSTR(QUAN,1,Q-1)||MULP||'+';
P_LOOP2: Q1 = Q1+Q;
      Q = INDEX(SUBSTR(QUAN,Q1),'+');
      IF Q ¬= 0 THEN
          DO;
              STRINGP = STRINGP||SUBSTR(QUAN,Q1,Q-1)||MULP||'+';
              GO TO P_LOOP2;
          END;
      CHARQ = LENGTH(QUAN);
      E_STRING = STRINGP||SUBSTR(QUAN,Q1,CHARQ-Q1+1)||MULP;
      RETURN;
END TIMES;


/*.....................SECTION 6 ......................*/


TERM: PUT SKIP(2);
      PUT SKIP LIST('NORMAL TERMINATION');

      /* END OF TREE MINIMIZATION PROGRAM */
END T_M_P;
```

MULTILEVEL-CIRCUIT DESIGN USING LOGIC TREES

by

HARRY LEE PUETT

B. S., Kansas State University, 1969

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1970

The logic tree described in this report is useful in multilevel-circuit design procedures. This logic tree employs only 2-input AND and OR gates and can mechanize any given Boolean function. The logic tree was derived from a recursion of the canonical sum-of-products form of general Boolean functions. The logic tree can represent a fixed Boolean function, or it can vary adaptively by adjusting the control inputs. This logic tree provides a standard circuit for computer-aided logic design applications and a structure for the minimization of Boolean functions.

The logic tree minimization procedure produces an economical multilevel representation of a Boolean function directly from the canonical form of a function. The application of a set of simple Boolean relationships is all that the logic tree minimization procedure requires to operate effectively.

A computer program which performs the logic tree minimization procedures is presented in the report. Other switching function minimization procedures are compared to the logic tree minimization procedure. The comparison of the minimization methods showed that the logic tree procedure does not have any of the disadvantages that the other methods have.