AN ABSTRACT OF THE THESIS OF

ROY LOUIS SCHIELE for the M.S. in Electrical Engineering
(Name) (Degree) (Major)

Date thesis is presented August 2, 1966

Title COMPUTER-AIDED MAJORITY LOGIC DESIGN

Abstract approved Redacted for Privacy
(Major professor)

The emergence of the three-input majority gate as a practical element for logical design has demanded a useful method of design using these elements. In order to facilitate the use of this gate, a digital computer program is presented to implement the design procedure.

Utilizing the truth table of a logical function, with "don't care" terms omitted, as the input, the program employs a unitizing method of function realization using three-input majority gates. The program complements and unitizes the table, and, using some fundamental theorems, reduces the table and selects the majority gate which gives maximum reduction. The reduction and gate selection processes continue until a final gate is obtained and the function is realized.

The result is an efficient and relatively fast method of logical design employing three-input majority gates.

The program was written in the FORTRAN II language for an IBM 1620 computer with 40K decimal digits of storage. In the form presented, it uses over 39K digits of storage.

COMPUTER-AIDED MAJORITY LOGIC DESIGN

by

ROY LOUIS SCHIELE

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1967

APPROVED:

Redacted for Privacy

Professor of Electrical and Electronic Engineering
In Charge of Major

Redacted for Privacy

Head of Department of Electrical and Electronic Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented August 2, 1966

Typed by Gwendolyn Hansen

ACKNOWLEDGEMENT

The author wishes to express his appreciation to Mr. L. N. Stone for his suggestions and guidance in the preparation of this thesis.

He would also like to thank Mr. J F. Engle for programming suggestions and his wife Sue Ann for encouragement while writing this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

COMPUTER-AIDED MAJORITY LOGIC DESIGN

I. INTRODUCTION

The present-day logic designer is no longer bound to the classical "and-or" or "nand-nor" logic as the only possibilities for a logical design. New methods of function realization, such as majority and threshold logic, have given him some much needed liberty in selecting the proper type of logic for a particular design.

It is important that the designer have at his disposal fast, efficient methods of logical design, utilizing several types of logic, so that he can produce several alternate designs in a reasonable amount of time. Then he would not be forced to use only one type of logic; instead, he could select the best logic for each function in a given machine. The result would be a design requiring less circuit elements with improved speed.

One possibility for a fast method of logic design would be a digital computer program to perform the logical design. This thesis will concern itself with the development of such a program for three-input majority logic design. The program would serve as a tool for the logical designer to improve his capabilities.

II. MAJORITY LOGIC

A three-input majority gate realizes the Boolean function:

$$M(A, B, C) = A \cdot B + B \cdot C + A \cdot C$$

Table I shows the truth table for this function, where the output is simply the majority of the inputs.

Table I. Truth table for the majority gate

INPUTS			OUTPUT
A	B	C	M(A, B, C)
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	1

The fact that the output must be the same as the majority of the inputs restricts the majority gate to an odd number of inputs. A number of practical circuits to realize majority functions have been presented in the literature (5, 6, 7, 11). The greater number of

these circuits use three-input majority logic, although some can easily be extended to use more inputs (5, 11).

The primary reason for the use of the three-input majority gate is that, in some cases, it gives a significant simplification in the logic circuit required to realize a function. However, a simplification is not guaranteed in every case. For instance, a logical function to compare binary numbers has the truth table shown in Table II.

To realize this circuit using three-input majority gates, three gates with nine inputs and a 1's and a 0's source are required. The logic diagram is represented in Figure 1, where "U" is the 1's source and "$\overline{U}$" is the 0's source.

This same circuit can be realized using the classical "and-or" logic with three gates and six inputs. The logic diagram using "and-or" logic is illustrated in Figure 2. Since, in general, "and" or "or" gates are less complex than majority gates, it is more practical to use the "and-or" logic in this particular case. The use of majority logic provides no simplification and in fact requires a more complex logic circuit.

However, in many cases majority logic does reduce the number of gates and inputs required to realize a function. Consider the truth table for a full adder represented in Table III. The "and-or" logic realization of the full adder is illustrated by the block diagram

Table II. Truth table to determine binary number correspondence

INPUTS		OUTPUT
A	B	F
1	0	1
0	1	1
0	0	0
1	1	0

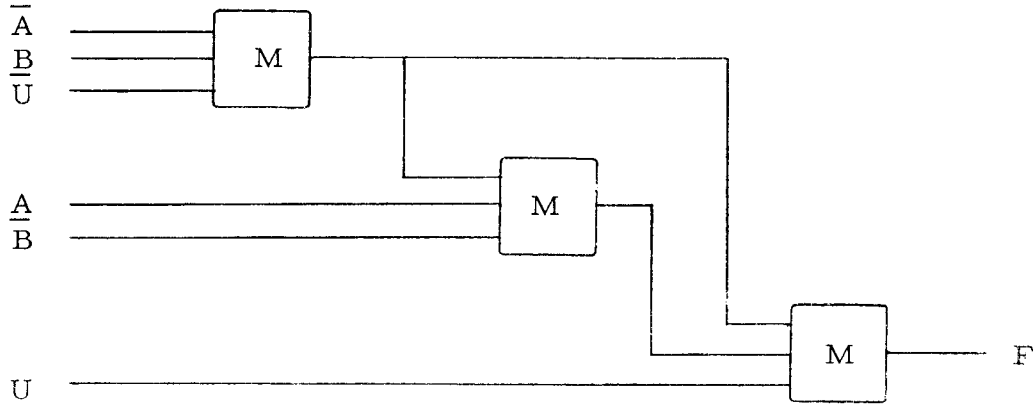

Figure 1. Majority gate realization of the function of Table II.

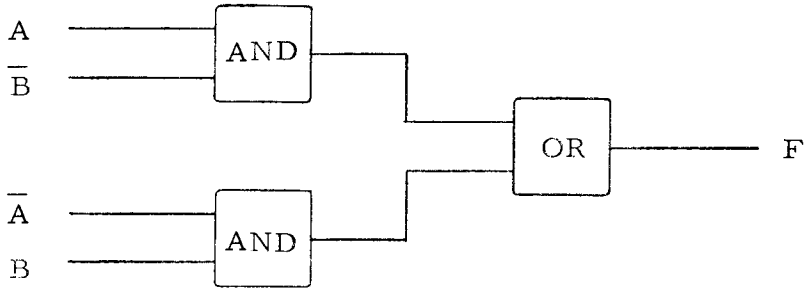

Figure 2. "And-or" realization of the function of Table II.

of Figure 3. This logic circuit requires four "and" gates, four "or" gates, and an inverter with a total of 16 inputs.

Table III. Truth table for a full adder

INPUTS			OUTPUTS	
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 4 shows the circuit realization of the full adder using three-input majority gates. For this logic circuit, four three-input majority gates with a total of 12 inputs are needed. Therefore it is evident that, in the case of the full adder, the use of the three-input majority gate gives a significant simplification.

The circuitry required for a majority gate is generally more complex than that required for "and-or" or "nand-nor" logic. However, the growing use of thin film and integrated circuits makes these more complex circuits practical both in size and in cost.

Since the majority gate is becoming practical, its application

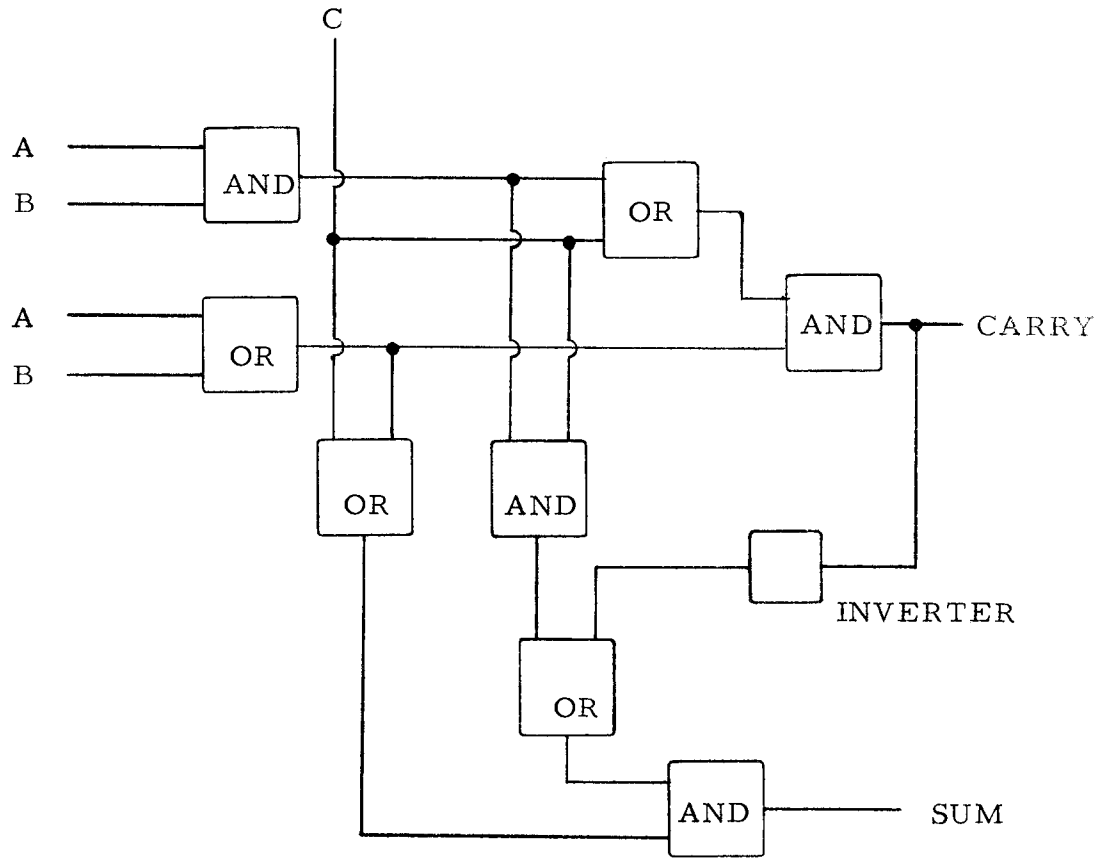

Figure 3. "And-or" realization of the full adder.

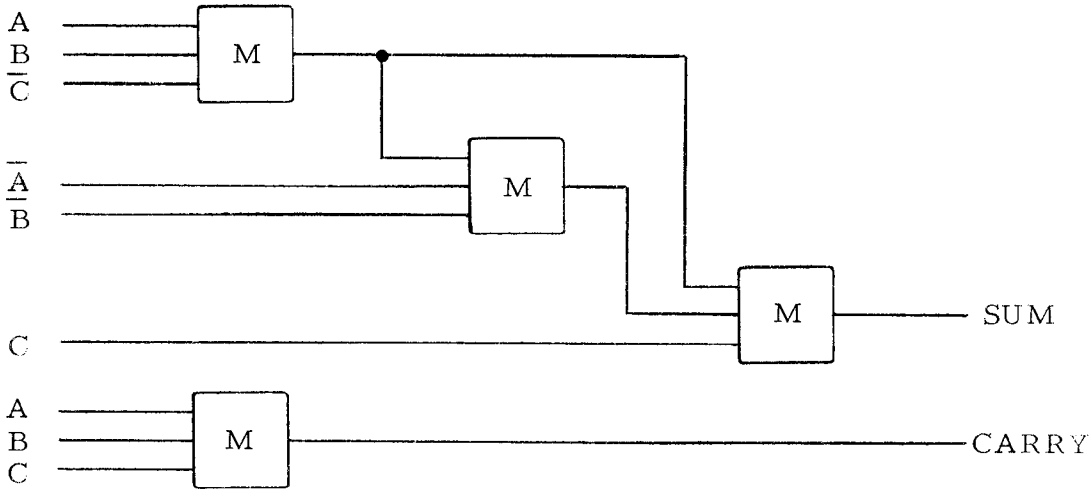

Figure 4. Majority gate logic circuit for the full adder.

in logic designs is being exploited. A good method of logical design utilizing majority gates is essential to the full development of this type of logic. This method should be relatively fast and applicable to logic functions of considerable size.

A number of methods of logical design using majority gates appear in the literature (1, 2, 3, 4, 8, 9, 10). The gate most commonly used in these methods is the three-input majority gate since it gives a good reduction in the logic circuit with ease of design. The complexity of these methods increases rapidly as the size of the function, or its corresponding truth table, being synthesized becomes large, and thus a practical and useful method of logic design using majority gates is not presently available for larger functions.

III. LOGICAL DESIGN PROGRAM

The repetitive nature of unitizing methods, and specifically the method presented by Akers and Robbins (1, 2, 3, 4), suggests a digital computer program approach to three-input majority gate logical design. This type of program would entail manipulation of the truth table of a given function to obtain a good logical design.

The program to be described is based on the method presented by Akers and Robbins (Appendix I). For clarity this method is divided into several parts and each part is treated separately. The over-all method is represented in Figure 5. The input to the program is the truth table for the function with any "don't care" terms omitted. The method utilizes these "don't care" terms by the fact that they are omitted.

Dividing and Complementing the Truth Table

The logic flow diagram to divide and complement the truth table is illustrated in Figure 6 on page 10. To facilitate later steps, the truth table is first divided into two subsets. All rows with a "1" output are placed in the first subset (the upper part of the truth table) and the rows with an output of "0" are placed in the second subset.

Next each variable, or column, in the truth table is

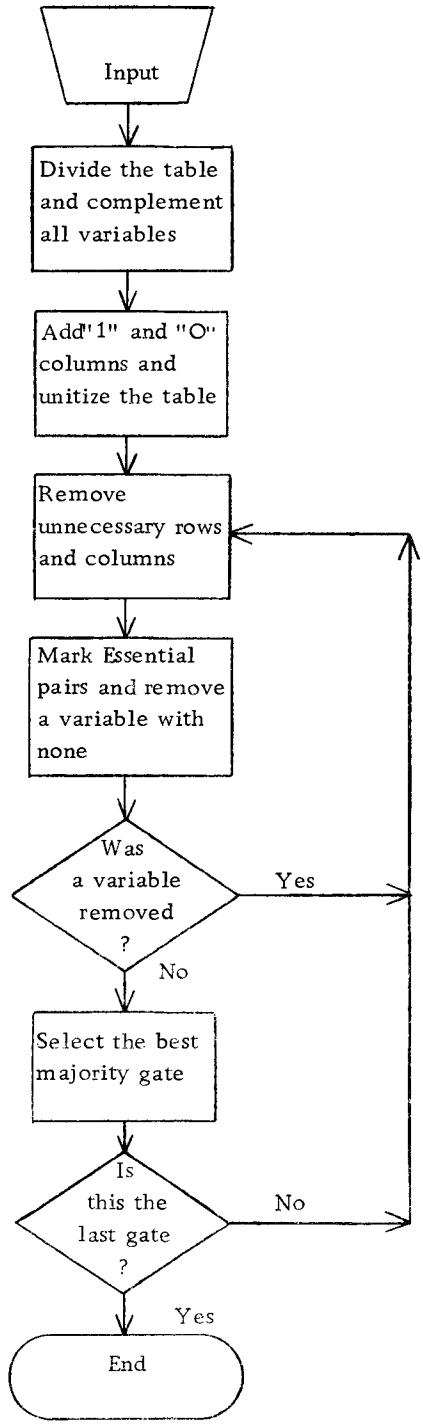

Figure 5. Process for logic design program.

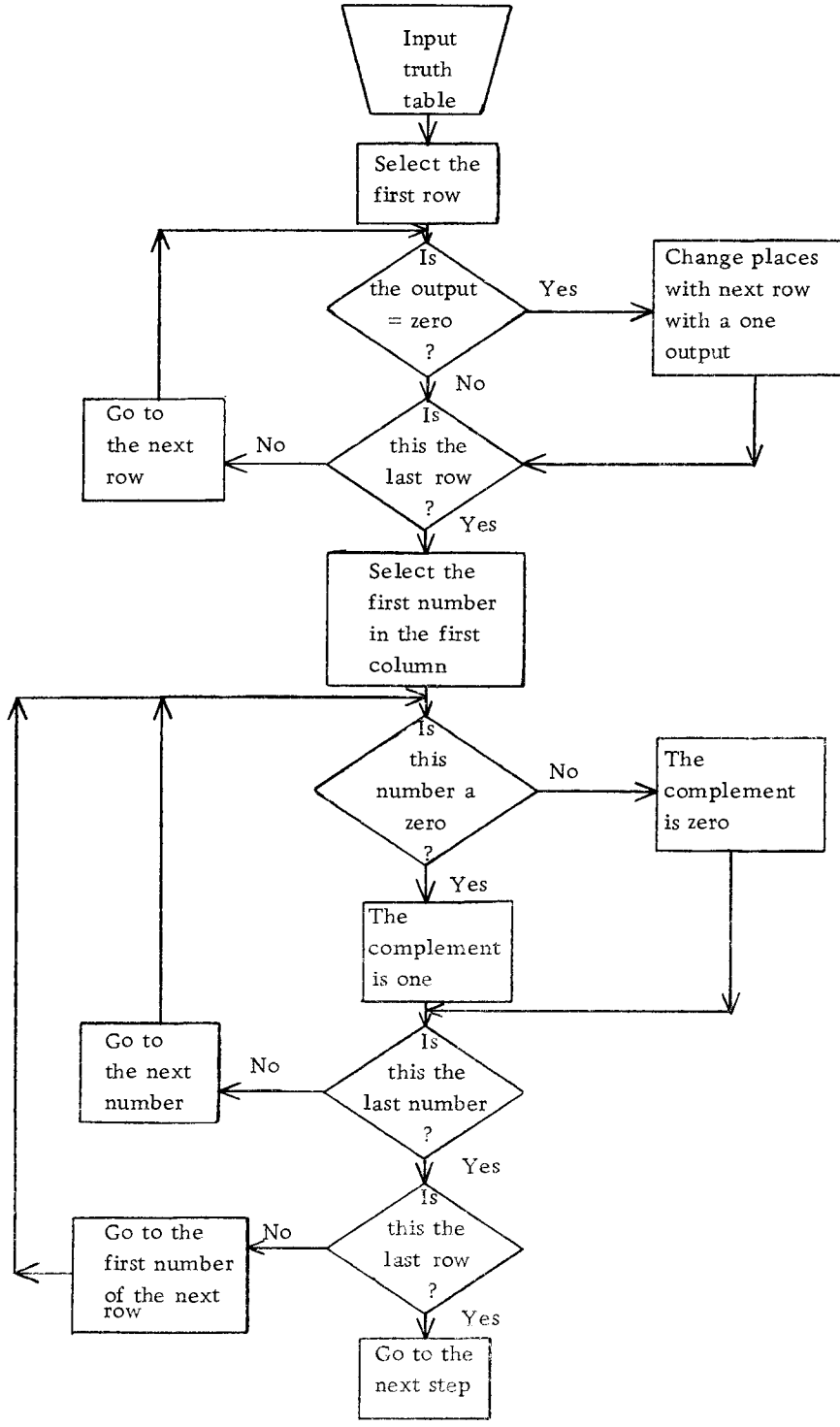

Figure 6. Dividing and complementing the truth table.

complemented and these complements are added to the truth table. This process doubles the size of the truth table. It is not always necessary to complement all variables, but any columns that are not required will be subsequently removed.

Adding 1's and 0's and Unitizing the Table

Once all the variables have been complemented, it is necessary to add a source of 1's and a source of 0's to guarantee that the function can be synthesized using three-input majority gates. These added variables are frequently not required, but for simplicity they are added here and will be removed later if not needed.

The next step is to unitize the truth table. This is simply a process of complementing any row that has a "0" for an output. The result is a unitized table which may be synthesized using three-input majority gates. The logic flow diagram to add a "1" and a "0" source and to unitize the truth table is shown in Figure 7.

Reducing the Truth Table

Once the unitized truth table is obtained, it can often be reduced by the removal of rows or columns. The logic flow diagram to remove rows and columns is illustrated in Figure 8 on page 13.

To remove rows, it is necessary to examine all possible pairs of rows. If one row has 1's in every column that another row has

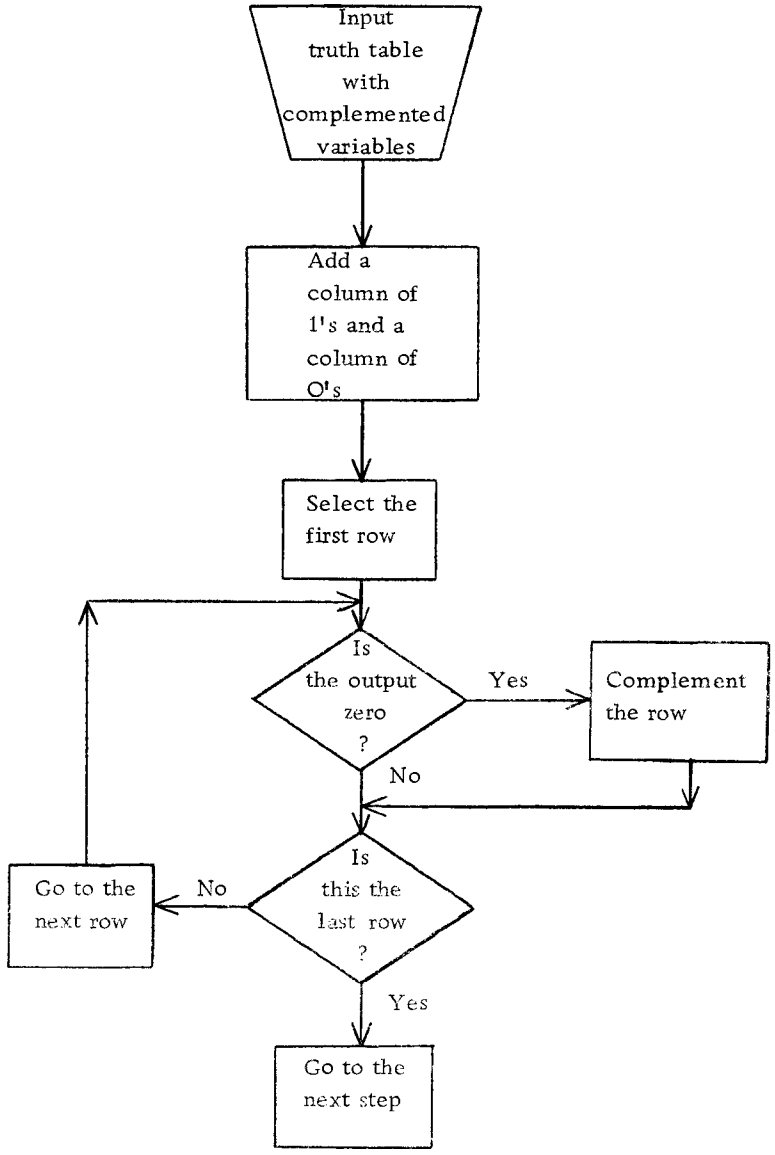

Figure 7. Adding columns of 1's and 0's and unitizing the table.

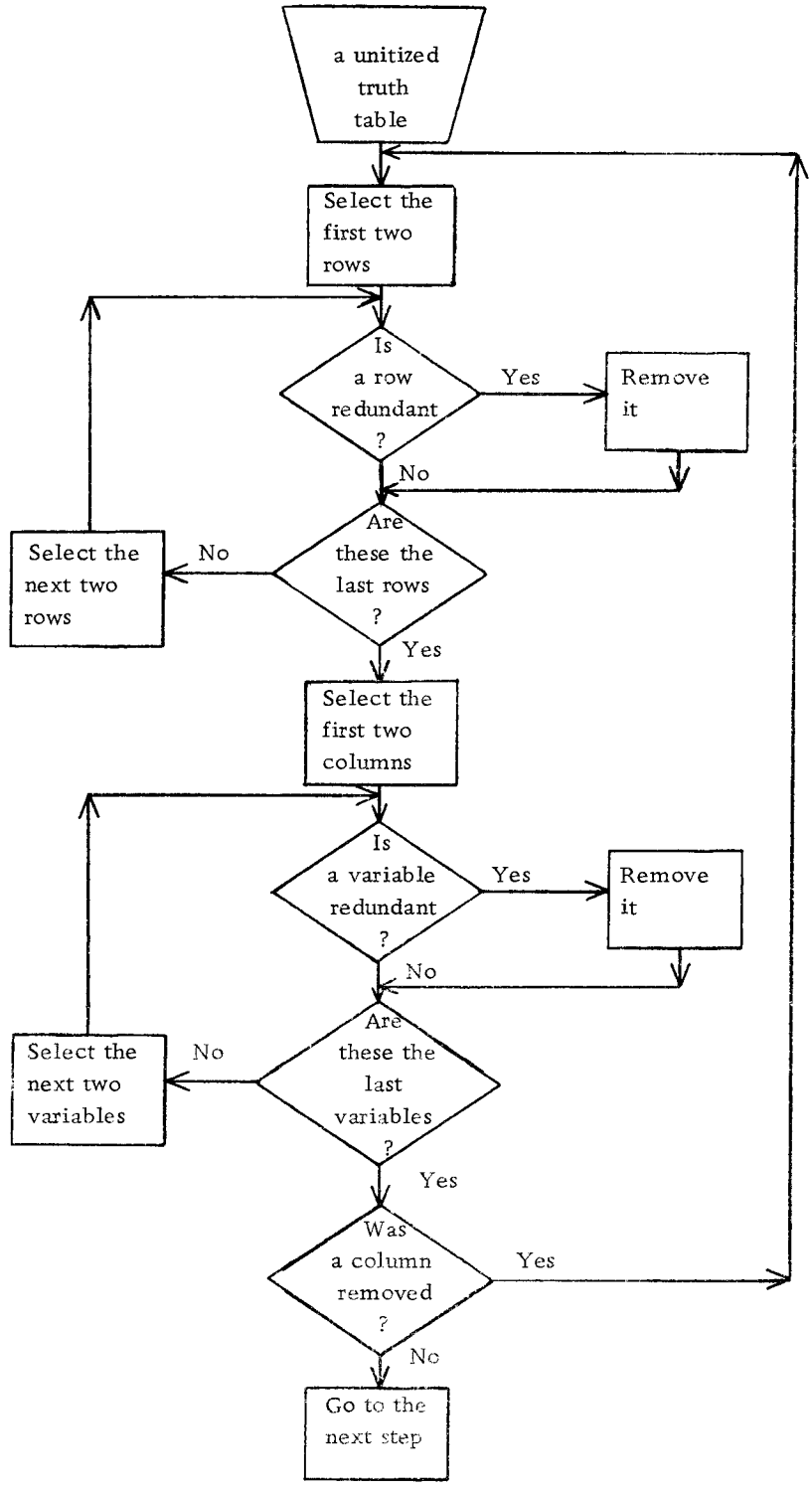

Figure 8. Removing rows and columns.

1's, the first row is redundant and may be removed. The entire table is searched and all redundant rows are removed. Then all possible pairs of columns are examined. If one variable has 1's in every position (row) that another variable has 1's, the second variable is redundant and can be removed from the truth table. Once again the entire table is searched and all redundant columns removed.

At the end of the process of column removal, if any variable was removed, it is necessary to go back to the step which removes the rows, since the removal of a column can cause new redundancies in the rows. Once no variables are removed in a pass through the process, the program proceeds to find essential pairs.

For any two rows in the truth table, there is at least one column that has 1's in each of the two rows. If only one column has 1's in the two rows, these 1's constitute an essential pair. Since any variable, or column, which does not contain an essential pair may be removed, it is necessary to search the table and locate these pairs. In the program they are marked by placing a negative sign before the "1" to obtain a "-1".

The program then searches the table to see if any variables contain no essential pairs and can thus be eliminated. Since, for most applications, the 1's and the 0's sources are not already available, it is usually desirable to remove either of these columns

first if a choice is to be made. The 1's and 0's sources are placed at the end of the table when they are added, so the program starts with the last column and searches toward the first in seeking variables with no essential pairs. Only one variable may be removed on a pass since the removal of two variables could leave a pair of rows with no essential pairs. Figure 9 illustrates the logic flow diagram to locate essential pairs and remove a variable with no essential pairs.

If a variable can be removed, it is deleted from the table and the essential pair marking is removed. The program then returns to remove any unnecessary rows and the process is repeated. Once all variables contain at least one essential pair, the next step is to select the best majority gate to reduce the table.

Selecting the Best Majority Gate

Once the fully reduced unitized table with essential pairs marked is obtained, the problem is to determine the majority gate which will give a maximum amount of reduction in the size of the truth table. The program searches the table to find a three-input majority gate that will eliminate as many variables as possible. The majority gate will eliminate a variable that does not have a "-1" (part of an essential pair) in any row where the majority gate output is a "0".

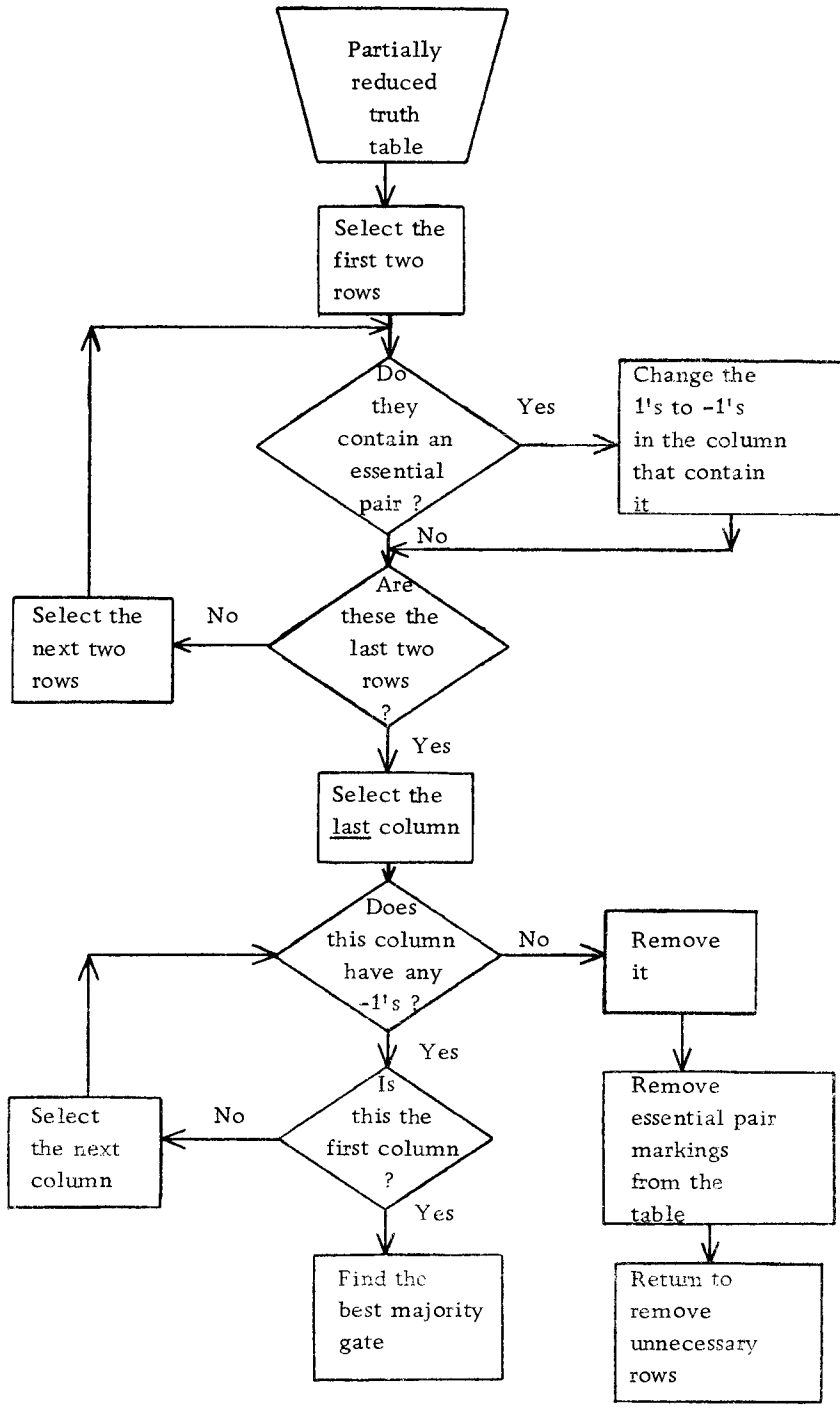

Figure 9. Marking essential pairs and removing a variable with none.

The program searches for a majority gate to eliminate three variables. If a majority gate to eliminate three variables does not exist, a gate to eliminate two, and then one, variable is sought. In each case the variables are removed, the majority gate is added to the table, and the program deletes essential pair markings and returns to remove unnecessary rows and further reduce the table.

After the variables are removed and the majority gate is added, if the table has only one column (the last majority gate added) the function has been realized and the process is complete. The output from the program is the majority gate found each pass of the program.

If a majority gate to eliminate even one variable does not exist, it is necessary to form a gate that does not eliminate any variables. This is done until a gate that will eliminate some variables exists. However, this normally results in a rather impractical design using three-input majority gates. For this reason, the process is terminated if a majority gate cannot be found to eliminate at least one variable. If the operator still desires a three-input majority gate realization of this function, he can add some gates to the truth table manually. It is necessary to add gates that have more 1's than some of the variables. After some gates are added manually, the truth table is again read into the computer and the function is realized. However, as stated above, this type of circuit normally requires a

large number of circuit elements.

The logic flow diagram to select the proper majority gate is shown in Figure 10. It is not necessary to remove the variables that can be removed by the majority gate, since these would be eliminated in reducing the table. However, they are removed here to increase the speed with which the program will run. A shorter modification of the program would not do this and, although some memory space would be conserved, time would be sacrificed.

Stage Reduction

For a good logical design, a relatively fast operating time is important. A gate has a certain propagation time, and this time multiplied by the number of stages gives a rough approximation of the delay through the logic circuit. A cower number of stages would give a lower propagation time and thus a better logical design.

If there were two possible majority gates which eliminated the same number of variables, it would be advantageous to use one that would possibly give a reduction in the number of stages. Thus to try to avoid using a majority gate that included, as an input, the majority gate that was added on the previous pass, would eliminate unnecessary building up of the stages.

In order to utilize this idea in the program, the table is first searched for a majority gate with the last column suppressed. Since

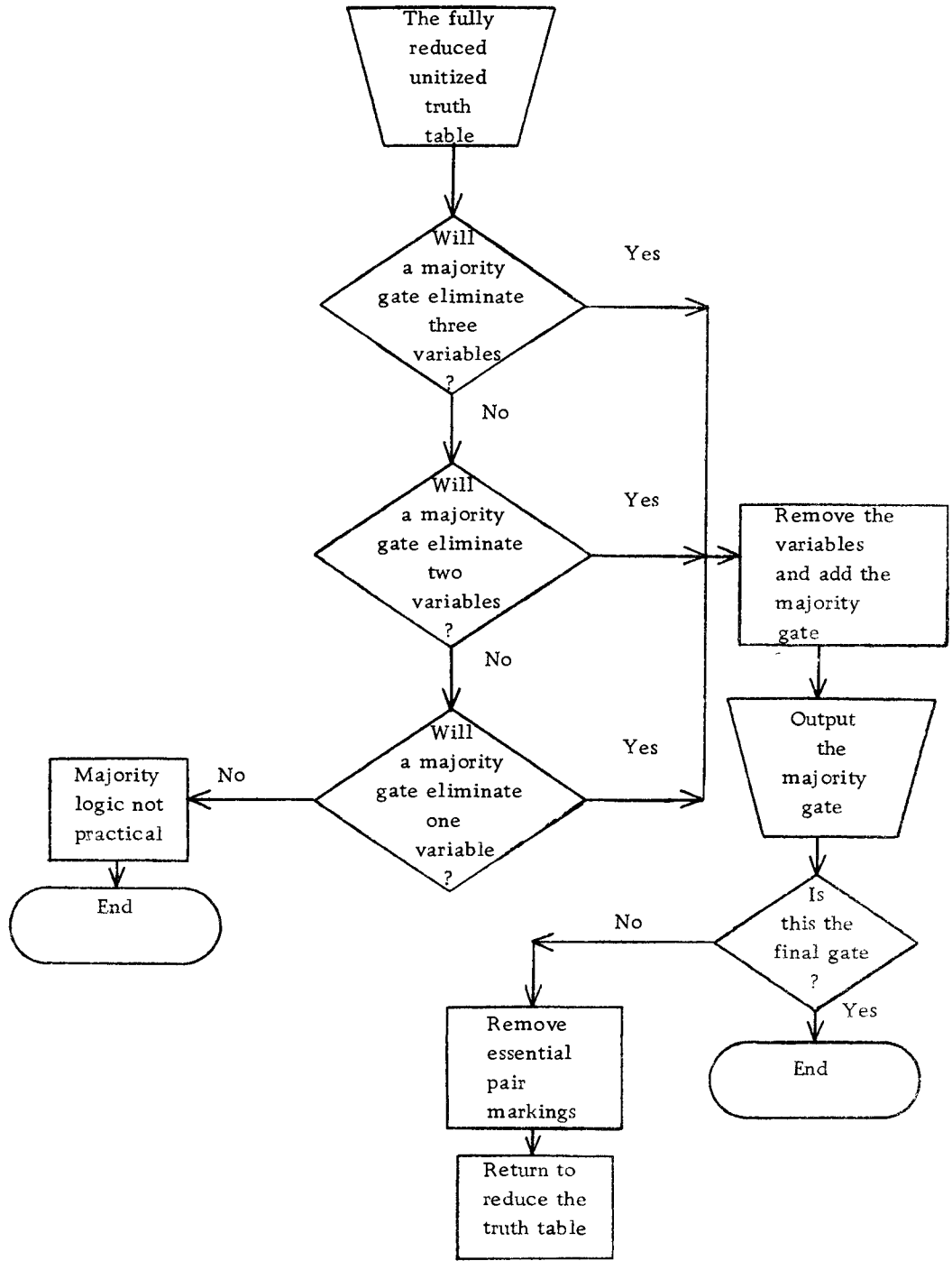

Figure 10. Selecting the best majority gate.

the majority gates are added onto the end of the table, this would avoid the use of the majority gate added on the last pass. If a gate is not found, the table is again searched including the last column. In this way, maximum reduction is still obtained, and where possible, some reduction in the number of stages may be obtained.

To see how this can help, consider the function whose truth table is shown in Table IV. This function was synthesized using three-input majority gates. First the problem was run without the stage reducing modification in the program, and the result was the three-stage logic circuit illustrated in Figure 11. Then the function was synthesized with the stage reducing modification in the program. The result was the two stage logic circuit shown in Figure 12. In this case, either circuit is correct and will realize the desired function, but the stage reduction gives an increase in speed of approximately one-third.

Table IV. Truth table

INPUT					OUTPUT
A	B	C	D	E	F
0	1	0	0	0	0
0	1	1	1	1	1
1	1	1	0	0	1
1	1	0	1	0	1
1	1	1	1	0	0

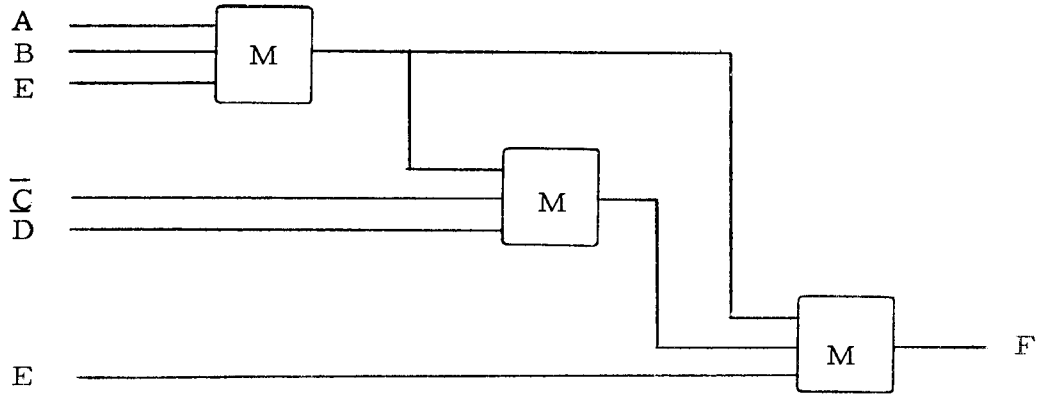

Figure 11. Logic circuit without stage reduction.

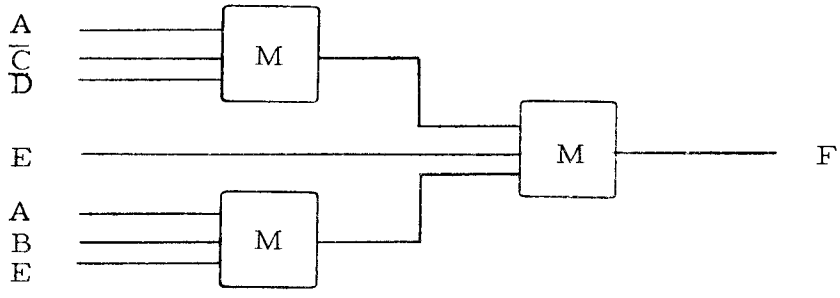

Figure 12. Logic circuit with stage reduction.

IV. COMMENT ON THE PROGRAM

The logical design program was written for an IBM 1620 computer with 40K decimal digits of core storage. The source deck language used was FORTRAN -II. This language was used to facilitate the understanding and modification of the program if desired. The SPS language was considered to conserve memory space, but the advantages of the FORTRAN-II language seemed to outweight the advantages of SPS.

Using the PDQ FORTRAN Processor C2, the program requires approximately 38,920 decimal digits of storage in its present form. A truth table with 6 variables and 40 rows is the maximum allowable with the 40K storage. The program requires $2k + 3$ columns for a truth table with k input variables, since it complements all variables, uses a "1" and a "0" input, and requires a storage column. It also requires $n + 1$ rows for a truth table with n rows since a storage row is used.

A larger memory would permit the synthesis of functions with much larger truth tables. The only change required in the program would be a modification of the DIMENSION statement to reserve space for the larger truth table, and an extension of the input READ statement. The only limiting factor on the size of truth table that can be synthesized is the running time of the program. Because

combinations are searched in the program, the running time is not a linear function of the size of the truth table. Running time is also dependent on the particular function being synthesized, so an accurate estimate of the running time can not be made. However, on the IBM 1620 problems were run in a relatively short time.

BIBLIOGRAPHY

1. Akers, Sheldon B. and Theodore C. Robbins. Logical design with three-input majority gates. Pt. 1. Computer Design 2:12-19. March 1963.

2. _____. Logical design with three-input majority gates. Pt. 2. Computer Design 2:20-23. April 1963.

3. _____. Logical design with three-input majority gates. Pt. 3. Computer Design 2:16-23. May 1963.

4. _____. Logical design with three-input majority gates. Pt. 4. Computer Design 2:20-27. June 1963.

5. Anand, S. K. Majority logic circuits. Master's thesis. Corvallis, Oregon State University, 1966. 109 numb. leaves.

6. Carr, W. N. and A. G. Milnes. Bias controlled tunnel-pair logic circuits. IRE Transactions on Electronic Computers 11:773-779. 1962.

7. Chow, W. F. Tunnel diode digital circuitry. IRE Transactions on Electronic Computers 9:295-301. 1960.

8. Lindman, R. A theorem for deriving majority logic networks within an augumented boolean algebra. IRE Transactions on Electronic Computers 9:338-342. 1960.

9. Miiller, H. S. and R O. Winder. Majority logic synthesis by geometric methods. IRE Transactions on Electronic Computers 11:89-90. 1962.

10. Miyata, Fusachika. Realization of arbitrary logical functions using majority elements. IEEE Transactions on Electronic Computers 12:183-191. 1963.

11. Sauer, W. A. Majority and threshold logic. Electronics 36:23-24. Nov. 29, 1963.

APPENDIX

APPENDIX I. MAJORITY LOGIC DESIGN

The following is the method for logical design using three-input majority gates used in the program for this thesis (1, 2, 3, 4).

The starting point of the process is a logically passive function, or a function that can be synthesized without inverters. A function $F(X_1, X_2, \ldots X_n)$ is said to be logically passive with respect to $(X_1, X_2, \ldots X_n)$ if and only if for any two input combinations, a_i and a_j, if $a_i \leq a_j$ then $F(a_i) \leq F(a_j)$. For binary numbers, $a_i \leq a_j$ if and only if a_j has units everywhere a_i does.

This definition lead to the following theorem:

Theorem I.

> A function $F(X_1, X_2, \ldots X_n)$ is logically passive with respect to $(X_1, X_2, \ldots X_n)$ if and only if F can be synthesized using only "and-or" gates.

A corollary follows:

Corollary I.

A function $F(X_1, X_2, \ldots X_n, 0, 1)$ is logically passive with respect to $(X_1, X_2, \ldots X_n, 0, 1)$ if and only if F can be synthesized using three-input majority gates.

Since $M(A, B, C) = A \cdot B + A \cdot C + B \cdot C,$

then $M(A, B, 1) = A \cdot B + A \cdot 1 + B \cdot 1 = A + B$

and $M(A, B, 0) = A \cdot B + A \cdot 0 + B \cdot 0 = A \cdot B$

Thus majority gates with the constants "0" and "1" reduce to "and-or" gates.

A logically passive function is obtained by complementing columns in the truth table so that the definition for a logically passive function is satisfied. Then the table can often be simplified by Theorem II.

Theorem II

Given the truth table for a logically passive function F, and any two rows a_i and a_j, where $a_i \leq a_j$, if $F(a_i) = F(a_j) = 1$, the a_j row may be removed and if $F(a_i) = F(a_j) = 0$, the a_i row may be removed.

The proof of this theorem follows from the definition of a logically passive function. To determine if the constants 0 or 1 or both are

required, we prove the following lemma:

Lemma I

If F is synthesized with only majority gates, then for any input a_i, $F(a_i) = \overline{F}(\overline{a}_i)$.

This is easily seen for a single majority gate and the general proof follows directly by induction on the number of gates. Functions having this property are called self-duals. Thus any function $F(X_1, X_2, \ldots, X_n)$ which has to be synthesized using only majority gates must be a self-dual function.

Theorem III

A logically passive function $F(X_1 X_2, \ldots, X_n)$ can be synthesized using only majority gates if and only if for any two inputs a_i and a_j with $F(a_i) = \alpha_i$ and $F(a_j) = \alpha_j$ there exists an X_k which has the valve α_i in a_i and α_j in a_j. Such a function will be called a logically passive self-dual (L. P. S. D.).

Since F is a logically passive function we consider only the cases where $F(a_i) = F(a_j) = 0$ and $F(a_i) = F(a_j) = 1$. Assuming the conditions not met and applying Lemma I we see that F is not a logically passive function. This shows that Theorem III is necessary.

Before showing that Theorem III is sufficient, we note that the table can be made an L.P.S.D., if it is not already, by addition of the constant "0" or "1", or both. The truth table of an L.P.S.D. can then be simplified by the following theorem.

Theorem IV.

> The table of an L.P.S.D. is unchanged if each a_i for which $F(a_i) = 0$ is replaced by $\bar{a}_i$ and the "0" in the F column replaced by a "1".

The proof of this theorem follows from the fact that it is a self-dual.

This transformation produces a unitized table and Theorem III states that every pair of rows must have a unit in common. Applying Theorem II, further reduction can often be obtained.

Theorem V

> Given the unitized table for an L.P.S.D., $F(X_1, X_2, \ldots X_n)$, if $X_i \leq X_j$, then X_i may be eliminated.

This follows from the fact that the only requirement on the unitized table was that each pair of rows have 1's in common in at least one column.

Given a fully reduced unitized table for a function, we construct a three-input majority gate whose output is equal to the majority of

the inputs. The column of values for this gate is added to the table as an additional variable, and the table is again reduced and the process repeated. The problem is to pick the best majority gate at each step. As a corollary to Theorem V, we find:

Corollary II

> If in a unitized table there exists three variables X_i, X_j, X_k such that no row has $X_i = 1$, $X_j = 0$, $X_k = 0$ then adding the majority gate $M(X_i, X_j, X_k)$ to the table will allow X_i to be eliminated.

Theorem VI

> A necessary condition for gate $M(X_i, X_j, X_k)$ to appear in the realization of a L.P.S.D., is that for each of the three variables, there exists a row in the unitized table where it is "0" and the other two are "1".

If this was not the case, $M(X_i, X_j, X_k) \leq X_i$ and the gate could be eliminated.

A necessary property of the unitized table is that every pair of rows have a unit in common. A method of synthesis can be based on those particular row pairs that have only one pair of 1's in common. Such a pair of 1's will be called an essential pair.

Once all essential pairs are located, if there were a variable in the table with no essential pair in its column, that column could be deleted from the table, since no pair of rows would depend on that variable for 1's.

Now for a variable to be eliminated by a majority gate it is only required that the gate have 1's in the rows where essential pairs are located in the variable. This follows from Theorem V or the preceeding paragraph.

A geometric method for synthesis is described, but this portion was not used in the program described in the thesis.

APPENDIX II. THE PROGRAM

```
C       LOGIC DESIGN USING THREE-INPUT MAJORITY GATES
C       FIRST INPUT CARD FORMAT 212, NUMBER OF COLUMNS
C       AND NUMBER OF ROWS
C       INPUT FORMAT IS 12, ONE ROW PER CARD
C       OUTPUT (F) IN COLUMN 2, INPUTS IN COLUMNS 4,
C       6, 8, ETC.
C       FOR INPUTS A, B, C ETC., A = 1, NOTA = 2,
C       B = 3, NOTB = 4, ETC.
C       DIVIDING THE TABLE INTO SUBSETS
        DIMENSION LA(41,15), LO(41), INPUT(15)
   24   READ 17, K, N
        K = K * 2
        KK = K - 1
        DO 18 I = 1, N, 1
       OREAD 19, LO(I), LA(I,1), LA(I,3), LA(I,5),
       ILA(I,7), LA(I,9), LA(I,11), LA(I,13), LA(I,15)
       ILA(I,9), LA(I,11), LA(I,13), LA(I,15)
   18   CONTINUE
        I = 1
        NT = N + 1
    5   IF (LO(I) - 1) 1, 2, 2
    2   IF (I - N) 3, 4, 4
    3   I = I + 1
        GO TO 5
    1   DO 6 J = 1, KK, 2
    6   LA(NT,J) = LA(I,J)
        LO(NT) = LO(I)
        12 = I + 1
   10   IF (LO(12) - 1) 7, 8, 2
    7   IF (12 - N) 9, 4, 4
    9   12 = 12 + 1
        GO TO 10
    8   DO 11 J = 1, KK, 2
        LA(I,J) = LA(12,J)
   11   LA(12,J) = LA(NT,J)
        LO(I) = LO(12)
        LO(12) = LO(NT)
        IF (12 - N) 12, 4, 4
   12   I = I + 1
        GO TO 5
C       COMPLEMENTING ALL VARIABLES
    4   DO 20 J = 1, KK, 2
        DO 20 I = 1, N, 1
```

```
      J2 = J + 1
      IF (LA(I,J)) 22, 22, 21
   22 LA(I,J2) = 1
      GO TO 20
   21 LA(I,J2) = 0
   20 CONTINUE
      DO 23 J = 1, K, 1
   23 INPUT(J) = J
C     ADDING A COLUMN OF ONES AND A COLUMN OF ZEROS
      KK = K + 1
      DO 100 I = 1, N, 1
  100 LA(I,KK) = 1
      INPUT(KK) = KK
      PRINT 28, KK
      K = KK
      KK = K + 1
      DO 110 I = 1, N, 1
  110 LA(I,KK) = 0
      INPUT(KK) = KK
      PRINT 27, KK
      K = KK
C     UNITIZING THE TRUTH TABLE
  104 I = 1
  130 IF (LO(I) - 1) 132, 133, 132
  133 IF(I - N) 134, 131, 131
  134 I = I + 1
      GO TO 130
  132 DO 135 J = 1, K, 1
      IF (LA(I,J) - 1) 136, 137, 137
  136 LA(I,J) = 1
      GO TO 135
  137 LA(I,J) = 0
  135 CONTINUE
      LO(I) = 1
      GO TO 133
C     REMOVING ROWS FROM THE TRUTH TABLE
  131 MJNO = 50
      NCT = 0
  116 II = 1
      12 = II + 1
   41 L = 0
      LL = 0
      NN = N - 1
      IF (II - N) 57, 58, 58
   57 J = 1
   56 M = LA(II,J) + LA(12,J)
      IF (M - 1) 46, 49, 46
   46 IF (J - K) 47, 42, 42
   47 J = J + 1
      GO TO 56
```

```
 42 LT = L + LL
    IF (LT - 0) 61, 61, 53
 49 IF (LA(II,J)) 51, 51, 52
 51 L = LA(I2,J)
    GO TO 53
 52 LL = LA(II,J)
 53 IF (L - LL) 60, 55, 64
 55 IF (I2 - N) 54, 40, 54
 40 II = II + 1
    I2 = II + 1
    GO TO 41
 54 I2 = I2 + 1
    GO TO 41
 60 IF(J - K) 47, 61, 61
 61 DO 67 IT = II, NN, 1
    IAI = IT + 1
    DO 67 J = 1, K, 1
 67 LA(IT,J) = LA(IAI,J)
    N = N- 1
    I2 = II + 1
    GO TO 41
 64 IF (J - K) 47, 65, 65
 65 IF (I2 - N) 700, 875, 875
875 N = N - 1
    GO TO 40
700 DO 68 IT = I2, NN, 1
    IAI = IT + 1
    DO 68 J = 1, K, 1
 68 LA(IT,J) = LA(IA1,J)
701 N = N - 1
    GO TO 41
C   REMOVING COLUMNS FROM THE TRUTH TABLE
 58 J = 1
    J2 = J + 1
    LLL = 0
165 L = 0
    LL = 0
    KK = K - 1
    IF (J - K) 150, 151, 151
150 I = 1
155 M = LA(I,J) + LA(I,J2)
    IF (M - 1) 152, 153, 152
152 IF (I - N) 154, 156, 156
154 I = I + 1
    GO TO 155
156 LT = L + LL
    IF (LT - 0) 157, 157, 158
153 IF (LA(I,J)) 159, 159, 160
159 L = 1
    GO TO 158
```

```
  160 LL = 1
  158 IF (L - LL) 161, 162, 166
  162 IF (J2 - K) 163, 164, 163
  164 J = J + 1
      J2 = J + 1
      GO TO 165
  163 J2 = J2 + 1
      GO TO 165
  161 IF (I - N) 154, 157, 157
  157 IF (J2 - K) 702, 703, 703
  702 DO 171 JT = J2, KK, 1
      JA1 = JT + 1
  171 INPUT(JT) = INPUT(JA1)
      DO 167 JT = J2, KK, 1
      JAI = JT + 1
      DO 167 I = 1, N, 1
  167 LA(I,JT) = LA(I,JA1)
      K = K - 1
      LLL = 1
      GO TO 165
  703 K = K - 1
      LLL = 1
      GO TO 164
  166 IF (I - N) 154, 168, 168
  168 DO 172 JT = J, KK, 1
      JA1 = JT + 1
  172 INPUT(JT) = INPUT(JA1)
      DO 169 JT = J, KK, 1
      JA1 = JT + 1
      DO 169 I - 1, N, 1
  169 LA(I,JT) = LA(I,JA1)
      K = K - 1
      J2 = J + 1
      LLL = 1
      GO TO 165
  151 IF (LLL - 1) 477, 116, 116
C     MARKING ESSENTIAL PAIRS
  477 I = 1
  181 I2 = I + 1
      IF (I - N) 170, 390, 390
  170 J = 1
      L = 0
  175 IF (LA(I,J)) 410, 391, 410
  410 IF (LA(I2,J)) 173, 391, 173
  173 L = L + 1
  391 IF (J - K) 174, 180, 180
  174 J = J + 1
      GO TO 175
  180 IF(L - 1) 176, 176, 177
  177 IF (I2 - N) 178, 179, 179
```

```
  178 I2 = I2 + 1
      GO TO 170
  179 I = I + 1
      GO TO 181
  176 J = 1
  184 IF (LA(I,J)) 490, 491, 490
  490 MJ1 = 1
      GO TO 492
  491 MJ1 = 0
  492 IF (LA(I2,J)) 493, 494, 493
  493 MJ2 = 1
      GO TO 495
  494 MJ2 = 0
  495 M = MJ1 + MJ2
      IF (M - 2) 182, 183, 182
  182 J = J + 1
      GO TO 184
  183 LA(I,J) = -1
      LA(I2,J) = -1
      GO TO 177
C     REMOVING A VARIABLE WITH NO ESSENTIAL PAIRS
  390 KK = K - 1
      J = K
  192 I = 1
  194 IF (LA(I,J)) 190, 191, 191
  190 IF (J - 1) 198, 198, 197
  197 J = J - 1
      GO TO 192
  191 IF (I - N) 193, 195, 195
  193 I = I + 1
      GO TO 194
  195 IF (J - K) 499, 200, 200
  499 DO 189 JT = J, KK, 1
      JA1 = JT + 1
  189 INPUT(JT) = INPUT(JA1)
      DO 196 JT = J, KK, 1
      JA1 = JT + 1
      DO 196 I = 1, N, 1
  196 LA(I,JT) = LA(I,JA1)
  200 K = K - 1
  411 DO 185 I = 1, N, 1
      DO 185 J = 1, K, 1
      IF (LA(I,J)) 115, 185, 185
  115 LA(I,J) = 1
  185 CONTINUE
      GO TO 116
C     SELECTING THE BEST MAJORITY GATE
C     TO REMOVE THREE VARIABLES
  198 IF (K - 3) 881, 902, 903
  903 K = K - 1
```

```
      KTEM = 0
      GO TO 904
  902 KTEM = 1
  904 KK = K - 1
      KKK = K - 2
      J = 1
  210 J2 = J + 1
  211 J3 = J2 + 1
  212 I = 1
  213 IF (LA(I,J)) 419, 420, 419
  419 MJ1 = 1
      GO TO 421
  420 MJ1 = 0
  421 IF (LA(I,J2)) 422, 423, 422
  422 MJ2 = 1
      GO TO 424
  423 MJ2 = 0
  424 IF (LA(I,J3)) 425, 426, 425
  425 MJ3 = 1
      GO TO 427
  426 MJ3 = 0
  427 M = MJ1 + MJ2 + MJ3
      IF (M - 1) 214, 217, 214
  214 IF (I - N) 215, 233, 233
  215 I = I + 1
      GO TO 213
  217 IF (LA(I,J)) 418, 416, 416
  416 IF (LA(I,J2)) 418, 417, 417
  417 IF (LA(I,J3)) 418, 214, 214
  418 IF (J3 - K) 218, 219, 219
  218 J3 = J3 + 1
      GO TO 212
  219 IF (J2 - KK) 220, 221, 221
  220 J2 = J2 + 1
      GO TO 211
  221 IF (J - KKK) 222, 223, 223
  222 J = J + 1
      GO TO 210
  223 IF (KTEM) 900, 900, 901
  900 K = K + 1
      GO TO 902
C     TO REMOVE TWO VARIABLES
  901 IF (K - 3) 881, 905, 906
  906 K = K - 1
      KTEM = 0
      GO TO 907
  905 KTEM = 1
  907 J = 1
      KK = K - 1
      KKK = K - 2
```

```
    I2 = 2
260 J2 = J + 1
261 J3 = J2 + 1
262 I = 1
    L = 0
    LL = 0
    LLL = 0
263 IF (LA(I,J)) 428, 429, 428
428 MJ1 = 1
    GO TO 430
429 MJ1 = 0
430 IF (LA(I,J2)) 431, 432, 431
431 MJ2 = 1
    GO TO 433
432 MJ2 = 0
433 IF (LA(I,J3)) 434, 435, 434
434 MJ3 = 1
    GO TO 436
435 MJ3 = 0
436 M = MJ1 + MJ2 + MJ3
    IF (M - 1) 264, 266, 264
264 IF ( I - N) 265, 271, 271
265 I = I + 1
    GO TO 263
266 IF (LA(I,J)) 450, 451, 451
451 IF (LA(I,J2)) 450, 452, 452
452 IF (LA(I,J3)) 450, 264, 264
450 IF (MJ1) 268, 268, 267
267 L = 1
    GO TO 271
268 IF (MJ2) 270, 270, 269
269 LL = 1
    GO TO 271
270 LLL = 1
271 IF (I2 - 1) 331, 331, 800
800 IF (L - LL) 276, 272, 279
276 IF (LL - LLL) 281, 281, 277
277 IF (I - N) 278, 275, 275
278 I = I + 1
    GO TO 263
272 IF (L - 0) 277, 277, 281
279 IF (L - LLL) 281, 281, 277
281 IF (J3 - K) 282, 283, 283
282 J3 = J3 + 1
    GO TO 262
283 IF (J2 - KK) 284, 285, 285
284 J2 = J2 + 1
    GO TO 261
285 IF (J - KKK) 286, 287, 287
286 J = J + 1
```

```
      GO TO 260
  287 IF (KTEM) 908, 908, 909
  908 K = K + 1
      GO TO 905
C     TO REMOVE ONE VARIABLE
  909 IF (K - 3) 881, 910, 911
  911 K = K - 1
      KTEM = 0
      GO TO 912
  910 KTEM = 1
  912 I2 = 1
      KK = K - 1
      KKK = K - 2
  331 IF (I - N) 332, 333, 333
  332 I = I + 1
      GO TO 263
  333 IF (L - LL) 334, 335, 334
  335 IF (LLL - 0) 334, 334, 336
  336 IF (J3 - K) 337, 338, 338
  337 J3 = J3 + 1
      GO TO 262
  338 IF (J2 - KK) 339, 340, 340
  339 J2 = J2 + 1
      GO TO 261
  340 IF (J - KKK) 341, 913, 913
  341 J = J + 1
      GO TO 260
  913 IF (KTEM) 914, 914, 881
  914 K = K + 1
      GO TO 910
C     REMOVES ONE VARIABLE
  334 IF (KTEM) 915, 915, 916
  915 K = K + 1
      KK = KK + 1
      KKK = KKK + 1
  916 KP = K + 1
      NCT = NCT + 1
      IF (NCT - 5) 880, 881, 881
  881 PRINT 882
      GO TO 400
  880 DO 440 I = 1, N, 1
      DO 440 JT = 1, K, 1
      IF (LA(I,JT)) 441, 440, 441
  441 LA(I,JT) = 1
  440 CONTINUE
      DO 345 I = 1, N, 1
      M = LA(I,J) + LA(I,J2) + LA(I,J3)
      IF (M - 2) 343, 344, 344
  343 LA(I,KP) = 0
      GO TO 345
```

```
  344 LA(I,KP) = 1
  345 CONTINUE
      MJ1 = INPUT(J)
      MJ2 = INPUT(J2)
      MJ3 = INPUT (J3)
      INPUT(KP) = MJNO + 1
      IF (L - 1) 346, 347, 347
  346 DO 350 JT = J, K, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JA1)
      DO 350 I = 1, N, 1
  350 LA(I,JT) = LA(I,JA1)
      GO TO 252
  347 IF (LL - 1) 348, 349, 349
  348 DO 352 JT = J2, KK, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JA1)
      DO 352 I = 1, N, 1
  352 LA(I,JT) = LA(I,JA1)
      GO TO 252
  349 DO 354 JT = J3, KKK, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JA1)
      DO 354 I = 1, N, 1
  354 LA (I,JT) = LA(I,JA1)
      GO TO 252
C     REMOVES TWO VARIABLES
  275 IF (KTEM) 917, 917, 918
  917 K = K + 1
      KK = KK + 1
  918 KP = K + 1
      DO 442 I = 1, N. 1
      DO 442 JT = 1, K, 1
      IF (LA(I,JT)) 443, 442, 443
  443 LA(I,JT) = 1
  442 CONTINUE
      DO 291 I = 1, N, 1
      M = LA(I,J) + LA(I,J2) + LA(I,J3)
      IF (M - 2) 288, 289, 289
  288 LA(I,KP) = 0
      GO TO 291
  289 LA(I,KP) = 1
  291 CONTINUE
      MJ1 = INPUT(J)
      MJ2 = INPUT(J2)
      MJ3 = INPUT(J3)
      INPUT(KP) = MJNO + 1
      IF (L - 1) 292, 295, 295
  292 DO 293 JT = J, K, 1
      JA1 = JT + 1
```

```
      INPUT(JT) = INPUT(JA1)
      DO 293 I = 1, N, 1
  293 LA(I,JT) = LA(I,JA1)
      J2 = J2 - 1
      K = K - 1
  295 IF (LL - 1) 296, 300, 300
  296 DO 297 JT = J2, K, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JA1)
      DO 297 I = 1, N, 1
  297 LA(I,JT) = LA(I,JA1)
      IF (L - 1) 252, 625, 625
  625 K = K - 1
  300 J3 = J3 - 1
  302 DO 306 JT = J3, KK, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JA1)
      DO 306 I = 1, N, 1
  306 LA(I,JT) = LA(I,JA1)
      GO TO 252
C     REMOVES THREE VARIABLES
  233 IF (KTEM) 919, 919, 920
  919 K = K + 1
      KK = KK + 1
      KKK = KKK + 1
  920 KP = K + 1
      DO 444 I = 1, N, 1
      DO 444 JT = 1, K, 1
      IF (LA(I,JT)) 445, 444, 445
  445 LA(I,JT) = 1
  444 CONTINUE
      DO 255 I = 1, N, 1
      M = LA(I,J) + LA(I,J2) + LA(I,J3)
      IF (M - 2) 237, 238, 238
  237 LA(I,KP) = 0
      GO TO 255
  238 LA(I,KP) = 1
  255 CONTINUE
      MJ1 = INPUT(J)
      MJ2 = INPUT(J2)
      MJ3 = INPUT(J3)
      INPUT(KP) = MJNO + 1
      DO 256 JT = J, K, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JA1)
      DO 256 I = 1, N, 1
  256 LA(I,JT) = LA(I,JA1)
      J2 = J2 - 1
      DO 257 JT = J2,KK, 1
      JA1 = JT + 1
```

```
      INPUT(JT) = INPUT(JA1)
      DO 257 I = 1, N, 1
  257 LA(I,JT) = LA(I, JA1)
      J3 = J3 - 2
      DO 258 JT = J3, KKK, 1
      JA1 = JT + 1
      INPUT(JT) = INPUT(JAI)
      DO 258 I = 1, N, 1
  258 LA(I,JT) = LA(I,JA1)
      K = K - 2
  252 MJNO = MJNO + 1
      PUNCH 16, MJNO, MJ1, MJ2, MJ3
      IF (K - 1) 400, 400, 411
  400 IF (SENSE SWITCH 9) 15, 24
   17 FORMAT (2I2)
   19 FORMAT (9I2)
  160 FORMAT (13HMAJORITY GATE, 13, 4H = (, I2, 1H,,
      II2, 1H,, I2, 1H)/)
 8820 FORMAT (33HMAJORITY LOGIC NOT REASONABLE FOR,
      114H THIS FUNCTION)
   28 FORMAT (5HINPUT, 13, 14H = UNITY INPUT/)
   27 FORMAT (5HINPUT, 13, 13H = ZERO INPUT/)
   15 END
```