Designing Six Variable Combination Logic Circuits with the TI-59

Summer 1981

Brian M. Ashford
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

Ashford, Brian M., "Designing Six Variable Combination Logic Circuits with the TI-59" (1981). Retrospective Theses and Dissertations. 533.
https://stars.library.ucf.edu/rtd/533

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
DESIGNING SIX VARIABLE COMBINATIONAL LOGIC CIRCUITS
WITH THE TI-59

BY

BRIAN M. ASHFORD
B.S.E., University of Central Florida, 1979

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the College of Engineering
at the University of Central Florida at Orlando, Florida

Summer Quarter
1981
ABSTRACT

A program has been written for the Texas Instrument's TI-59 hand-held calculator implementing the Quine-McCluskey minimization method for logic circuit design. This program is contained on multiple magnetic cards and provides the user with the capability for combinational logic minimization of circuit design problems containing up to six variables.
ACKNOWLEDGMENT

Since the work resulting in this paper has involved many people, I would like to acknowledge those most helpful. A great debt is owed to my advisor and committee chairman, Dr. Fred O. Simons for his assistance throughout my research project. With his assistance I have been able to conquer many problems encountered during the course of this program.

Thanks is also given to my other committee members: Dr. Richard C. Harden and Dr. Ernest E. Erickson for their help and constructive comments during my entire program.

Finally, and most of all, I would like to thank my wife, Rosemaree, and my parents Jack Ashford and Marion Ashford. They have been wonderful and enthusiastic supporters all throughout my graduate studies. To my parents and especially my wife, I am dearly grateful.
# TABLE OF CONTENTS

| LIST OF FIGURES | ............................................... | v  |
| INTRODUCTION | ............................................... | 1  |
| CHAPTER I. REVIEW OF THE QUINE-McCLUSKEY ALGORITHM | ............................................... | 5  |
| CHAPTER II. ALGORITHM ADAPTATION TO THE TI-59 | ............................................... | 7  |
| CHAPTER III. A COMBINATIONAL LOGIC DESIGN EXAMPLE | ............................................... | 15 |
| CHAPTER IV. CONCLUSION | ............................................... | 20 |
| APPENDIX A | ............................................... | 21 |
| APPENDIX B | ............................................... | 27 |
| LIST OF REFERENCES | ............................................... | 36 |
LIST OF FIGURES

1. SELECTION/DATA ORGANIZATION - Card 1/side 1 ............. 12
2. SELECTION/DATA ORGANIZATION - Card 1/side 2 ............. 12
3. PASS 1 - CARD 2 and PASS 2 - CARD 3 ..................... 13
4. PASS 3 - CARD 4 .............................................. 14
5. SEQUENCE (VCLR Description) ................................ 28
6. IFTHENELSE (VCLR Description) ............................. 28
7. IFTHENELSE CONSTRUCT WITH NULL PATH (VCLR Description) ... 29
8. NESTED IFTHENELSE (VCLR Description) .................... 29
9. DOCASE (VCLR Description) .................................... 31
10. NESTED IFTHENELSE EQUIVALENT TO DOCASE (VCLR Description) .................................................. 32
11. DOWHILE (VCLR Description) ................................. 32
12. DOUNTIL (VCLR Description) ................................. 34
13. DOLOOP (VCLR Description) .................................... 34
INTRODUCTION

In October 1952 W. V. Quine proposed the following Definitions and Theorems in order to solve the problem of reducing a normal formula to its simplest normal equivalent.

This normal formula is defined as any alternation of fundamental formulas where a fundamental formula is the conjunction of literals. In the following text, the Greek letter 'ξ' will refer to any literal, 'φ', 'ψ', 'χ' will refer to any fundamental formula and 'Φ' and 'Ψ' will more generally refer to any normal formula.

DEFINITIONS AND THEOREMS

Definitions - \( \phi \) will be said to subsume \( \psi \) if and only if all the literals whereof \( \psi \) is a conjunction are among the literals whereof \( \phi \) is a conjunction. \( \phi \) will be called a prime implicant of \( \psi \) if and only if \( \phi \) implies \( \psi \) and subsumes no shorter formula which implies \( \psi \). \( \phi \) will be called a completion of \( \chi \) with respect to \( \psi \) if and only if \( \phi \) subsumes \( \chi \) and contains all letters of \( \psi \) and no others.

Theorem 1 - Any simplest normal equivalent of \( \phi \) is an alternation of prime implicants of \( \phi \).

Theorem 2 - No prime implicant of \( \phi \) contains letters foreign to \( \phi \).

Theorem 3 - If \( \phi \) is a developed normal formula and contains all letters of \( \psi' \), then \( \psi' \) implies \( \phi \) if and only if all completions of \( \psi' \) with respect to \( \phi \) are clauses to \( \phi \).
Theorem 4 - \( \psi \) is a prime implicant of a developed normal formula \( \phi \) if and only if all letters of \( \psi \) are among those of \( \phi \) and all completions of \( \psi \) with respect to \( \phi \) are clauses of \( \phi \) and there is no shorter formula \( \psi \), subsumed by \( \psi \) such that all completions of \( \psi \) with respect to \( \phi \) are clauses of \( \phi \).

Theorem 5 - If \( \psi \) is a simplest normal equivalent of a developed normal formula \( \phi \), then each clause of \( \phi \) subsumes a clause of \( \psi \).

Theorem 6 - The only redundant normal formulas which are valid are 'pvp', 'qvq', etc.

Theorem 7 - If no two of \( \phi_1, \ldots, \phi_n \) have letters in common, and \( \phi \) is a prime implicant of \( \phi_1 \lor \cdots \lor \phi_n \), then \( \phi \) contains letters exclusively of \( \phi_i \); for some \( i \).

Theorem 8 - If \( \phi_1 \lor \cdots \lor \phi_n \) (\( n > 1 \)) is redundant and no two of \( \phi_1, \ldots, \phi_n \) have letters in common, then any simplest normal equivalent of \( \phi_1 \lor \cdots \lor \phi_n \) will be of the form \( \psi_1 \lor \cdots \lor \psi_n \) where \( \psi_1, \ldots, \psi_n \) are equivalent respectively to \( \phi_1, \ldots, \phi_n \).

Quine was able to develop a mechanical routine by which a normal formula would be reduced to a redundant equivalent where there were no superfluous clauses and none of the clauses had superfluous literals. This was accomplished by testing all clauses for implication in the rest of the formula, delete it, and when all clauses had been tested in this manner, then use the same procedure on all subclauses. After the testing was complete, a redundant formula would be achieved. Through the use of prime implicant tables and their reduced versions as covered in Theorems 1-5, the simplest normal equivalent was found but through a more tedious method than the first one mentioned for locating and eliminating redundancies. Quine admitted that there must be a quicker and easier adaptation of his
ideas in achieving the simplest normal equivalents but at the time, he did not see a way. Later in November 1956, E. J. McCluskey adapted Quine's basic ideas to logic reduction so that they could be easily implemented on a digital computer.

In November 1953 M. Karnaugh presented another approach to logic minimization through a graphical approach using minimizing charts or maps. Karnaugh took the minimizing chart developed at the Harvard Computation Laboratory and expanded the initial design to handle multi-input variables. Taking advantage of the human facilities for recognizing geometric patterns, the methods laid out by Karnaugh allowed the designer several short cuts in logic minimization by using the map (cube) representation for all possible combinations of a multiple-input circuit. This method presented a relatively easy and quick to learn procedure for logic designers; however, as the variable inputs increased, the mental gymnastics required for simplification became increasingly difficult. Karnaugh realized this and so stated that as the cubes became more complex (six variables and greater), other methods could possibly be better and simpler to use. When non-trivial problems did arise with less than six variables, the profitability was easily seen due to the inherent speed and flexibility of Karnaugh's map method. Because of this, as a desk-top aid for the logic designer in handling small to medium sized input systems, Karnaugh's map method has proven to be extremely successful.

With the two methods available for logic design, Karnaugh Maps and the Quine-McCluskey Technique, the designer now has two powerful tools for digital design minimization. In present day use the
Karnaugh Map is very easy and simple to use for problems with up to four variables. Beyond this number it becomes extremely difficult to be assured that one has the simplest minimization. The Quine-McCluskey technique was formulated so that the minimization technique could be implemented on a digital computer. With the Quine-McCluskey technique, problems with greater than four variables can be handled easily and the simplest realization of the function is always obtained. However, implementing this technique by hand is long and cumbersome. The use of these two methods leaves a gap for the designer involving from four to six variables. The Karnaugh Map will not necessarily guarantee the simplest realization and the Quine-McCluskey technique is a very long and tedious process. To bridge this gap, a program for use on the TI-59 hand calculator has been developed using the Quine-McCluskey minimization method.

A set of algorithms has been designed and implemented, adapting the Quine-McCluskey method to the TI-59. The developed algorithms and requirements are covered later in this paper, and a listing of the program is shown in Appendix A. Program utilization is demonstrated by a five variable logic design example. Afterwards, conclusions on the uses and limitations of the program are presented.
I. REVIEW OF THE QUINE-McCLUSKEY ALGORITHM

Although the Karnaugh-Map is a powerful tool in combinational logic design, there is no specific, straightforward procedure in obtaining the smallest possible set of products. A tabular method developed by W. V. Quine and E. J. McCluskey guarantees the best second order or level realization. This method is an organized approach in searching for all possible combinations of zero-cubes into higher dimension cubes. These cubes correspond to the Karnaugh Map in the following manner; zero cubes are equivalent to single squares, one-cubes are equivalent to adjacent squares and so forth on the K-Map. Selection of minimal combination of these cubes is realized for the desired function.

At this point the Quine-McCluskey technique will not be formally discussed, however, a simple outline of the basic steps will be presented. For a more in depth discussion of the Quine-McCluskey technique, the Hill, Peterson text and the Dietmeyer text found in the Bibliography section cover the subject amply.

The initial step in a Quine-McCluskey solution is the listing of minterms. From these minterms, the one-cubes are determined by combining two zero-cubes (minterms) which are identical except for one variable. The comparison process is repeated between all minterms, and if any fail to combine, these are then considered prime implicants.
The next step is to make the same comparison as above, but now search for two cubes. Two-cubes are two one-cubes that have been combined by the same criteria and again differ by just one variable. If any one-cubes do not combine, they are prime implicants. This procedure will keep repeating as the cubes become larger, until no more cubes can be created.

The resulting prime implicants are the minimum combinations required to realize the function.
II. ALGORITHM ADAPTATION TO THE TI-59

In adapting the Quine-McCluskey technique to the TI-59, many problems had to be overcome. The first and most important was memory requirements for number storage. In a six variable logic problem the memory required was found to exceed the 100 register capability on the TI-59. To allow the Quine-McCluskey technique to be used on the TI-59, the following initial steps had to be used.

1. If the total number of minterms is greater than 32, then replace the minterms with maxterms and use an inverter on the output.

2. Use number packing in the registers. Up to four numbers are placed within a register.

3. The storage of minterm combinations (next higher cubes) will start at a low number in memory and increment, and the storage of possible prime implicants will start at high memory and decrement.

The utilization of the above steps reduced the required memory to less than 65 registers.

Once the memory requirement problems were solved, the next problem was to develop algorithms for memory utilization. Again size was the major constraint on the program development, so speed was
sacrificed as a trade-off. The resulting program was designed for use on four cards with five sections.

The card design was set-up for minimum card usage. The first card, titled Selection/Data Organization, contains on side 1 the initial memory, set-up and initialization, and on side 2 another memory set-up and initialization for a later part in the program. Card 2 contains on both sides PASS 1; card 3 contains on both sides PASS 2; and card 4 contains on the first side PASS 3; and side 2 is not used. This allows storage of the program with over 1000 steps to be contained on only four cards.

The following is a detailed discussion of the five program sections with structured flow diagrams of all sections shown in Figures 1 - 5. (A discussion on the following flow diagrams is covered in Appendix B).

1. **SELECTION/ DATA ORGANIZATION**
   Card 1/Side 1

   - Initially the program steps and memory registers are changed to accommodate the requirements of both PASS 1 and PASS 2. All minterms or maxterms as the case may be are then entered at this point of the program. It is up to the designer to place the numbers into groups that differ from each other by the number of bits set in the binary form, i.e., for a six variable input:

<table>
<thead>
<tr>
<th>DECIMAL VALUE</th>
<th>BINARY VALUE</th>
<th>RESPECTIVE GROUPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000001</td>
<td>group 1</td>
</tr>
<tr>
<td>3</td>
<td>000011</td>
<td>group 2</td>
</tr>
<tr>
<td>4</td>
<td>000100</td>
<td>group 1</td>
</tr>
<tr>
<td>15</td>
<td>001111</td>
<td>group 4</td>
</tr>
<tr>
<td>33</td>
<td>100001</td>
<td>group 2</td>
</tr>
</tbody>
</table>
When these groups of numbers are entered into the calculator, they are then packed in memory four numbers to a register for conservation of space. Pointers are also set up for the beginning locations of each group and the number of minterms/maxterms that are within each of these groups. These pointers and group quantities will be used in PASS 1 when comparisons are made between these new groups.

2. PASS 1
   Card 2

   - At this point the zero-cubes (the groups created in the previous section) are compared for the possible formation of one-cubes or the determination of zero-cube prime implicants. By using both the pointers and group quantities for the first two groups, the following comparison is made for each number in group one versus group two.

   \[
   \text{GROUP 1 (X1)} \quad X2 - X1 = 1, 2, 4, 8, 16, 32 \\
   \text{GROUP 2 (X2)}
   \]

   If the test is true then the numbers X1 and X2 are stored in a higher location in memory as a one-cube with two one-cubes per memory location. These combinations are then grouped according to the way they are formed. Group 1 and group 2 zero-cubes will form group 1 one-cubes, group 2 and group 3 zero-cubes will form group 2 one-cubes and so forth. If a number in group one is not used to form a one-cube then that number is stored in decrementing registers at #79 and is considered a prime implicant.

   When all numbers in group one have been tested, the same procedures happen with group two numbers versus group three numbers and so forth until all groups have been tested. Again pointers and quantities for the created one-cube groups are maintained for the next section of comparisons in PASS 2. At this time all the prime implicants found in decreasing memory locations starting at #79 until a zero has been found must be physically recorded for the final solution.

3. PASS 2
   Card 3

   - This section of the program compares one-cubes for the possible formation of two-cubes or the determination of one-cube prime implicants. By using both the pointers and group quantities supplied in PASS 1, the first two groups are
compared as previously done in PASS 1 but the following test is used instead.

GROUP 1 \((X_1, Y_1)\) \(Y_1 - X_1 = Y_2 - X_2\) and

GROUP 2 \((X_2, Y_2)\) \(X_2 - X_1 = 1, 2, 4, 8, 16, 32\)

If the test is true then the numbers \(X_1, Y_1\) and \(X_2, Y_2\) are stored in another location in memory as two-cubes with one two-cube per memory location. These combinations are again grouped as previously done according to the way that they are formed. Group 1 and group 2 one-cubes will form the group 1 two-cubes, group 2 and group 3 one-cubes will form the group 2 two-cubes and so forth. If a one-cube is not used to form a two-cube then that one-cube is stored in decreasing memory starting at location #79. When all numbers in group one have been tested, the same procedures happen with the group 2 one-cubes and the group 3 one-cubes and continues until all one-cube groups have been tested. Again pointers and quantities for the created two-cube groups are maintained for the next section of comparisons in PASS 3. At this time all the prime implicants found in decreasing memory locations starting at #79 until a zero has been found must be physically recorded for the final solution.

- At this point the memory allocation within the calculator is changed to allow less room for program storage required by PASS 3. The memory size for data storage at the same time is increased to accommodate the increase in memory required by the three-cubes that are created in PASS 3.

- In this final section of the program all the two cubes in the groups created in the previous section are compared for the possible formation of three-cubes or the determination of two-cube prime implicants. By using both the pointers and group quantities for the first two groups, the following comparison is made for each two-cube in group 1 versus each two-cube in group 2.

GROUP 1 \((W_1, X_1, Y_1, Z_1)\) \(W_1 - X_1 = W_2 - X_2\) and

GROUP 2 \((W_2, X_2, Y_2, Z_2)\) \(W_2 - W_1 = 1, 2, 4, 8, 16, 32\)

If this test is true then the numbers \(W_1, X_1, Y_1, Z_1\) and \(W_2, X_2, Y_2, Z_2\) are stored in two consecutive memory locations as a three-cube starting
at location #14 in memory. If a two-cube is not used to form a three-cube then that number is stored at location #84 in memory to be recorded as a prime implicant when PASS 3 is completed.

At this point the program is completed and all newly created three-cubes are to be recorded from memory. Starting at memory location #84 and decreasing in order, more prime implicants will be found and are to be recorded. When the list of both three-cubes and prime implicants is complete, any redundancies are to be eliminated and the result that is left is the minimal realization of the desired function.
Change Memory to 319.79, set all memory pointers

Recall storage register, shift by 2, store new minterm

Ready for new group of minterms?

Y

NULL

N

All minterms inputted?

Y

N

Calculate total minterm number, reinitialize some of the pointers

Advance o-cube pointers, increment storage register

STOP

Figure 1. Selection/Data Organization
Card 1/Side 1

Change memory to 239.89, store new values and reinitialize register pointers

STOP

Figure 2. Selection/Data Organization
Card 1/Side 2
Store new values and reinitialize register pointers

Obtain number in first group for algorithm to compare

<table>
<thead>
<tr>
<th>Y</th>
<th>Do they satisfy the equation?</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Combine two groups of nos. and store in memory

<table>
<thead>
<tr>
<th>Y</th>
<th>All of second group tested?</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Has first group of numbers matched any second group of numbers?

<table>
<thead>
<tr>
<th>Y</th>
<th>Store first group number as a prime implicant</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>NULL</td>
</tr>
</tbody>
</table>

All of first group of numbers compared?

<table>
<thead>
<tr>
<th>Y</th>
<th>Increment pointers, get next first and second no. groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Finished comparing all numbers?

<table>
<thead>
<tr>
<th>Y</th>
<th>STOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Figure 3. Pass 1 - Card 2 and Pass 2 - Card 3
Store new values and reinitialize register pointers

Obtain number in first group for algorithm to compare

Obtain number in second group for algorithm to compare

Do they satisfy the equation?

Y

N

Combine two groups of nos. and store in memory

NULL

All of second group tested?

Y

NULL

N

Has first group number matched any second group number?

Y

NULL

N

Store first group number as a prime implicant

All of first group numbers compared?

Y

N

Increment pointers, get next first and second no. groups

Finished comparing all numbers?

Y

NULL

N

STOP

Figure 4. Pass 3 - Card 4
III. A COMBINATIONAL LOGIC DESIGN EXAMPLE

A simulated six variable problem involving the following minterms 1, 2, 3, 4, 5, 6, 7, 9, 11, 13, 15, 28, 35 is presented to demonstrate the use of the developed program.

Step 1-A) The number of minterms is less than 32 so changes for maxterms are not needed.

   B) Press key 1; read SELECTION (side 1); and press key A. When this is done, enter the minterms in the proper groups as described above, pressing key B after each minterm and key C after each group. (For the first group, press key 1; press key B; press key 2; press key B; press key 4; press key B; press key C. Continue this process until all numbers have been entered).

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
<th>Group 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
</tbody>
</table>

C) When all minterms have been entered, press key D.

NOTE: If there are numbers 0 and/or 63, they are to be entered in the following fashion. For the number 0, press key 0 and then press key A'. For the number 63, press key 6; press key 3 and then press B'. This will initialize certain flags for these two minterms.

Step 2-A) Press key 1; read PASS 1 (side 1); press key 2; read PASS 1 (side 2); and then press key A.

  B) After PASS 1 is completed, the prime implicants are stored in memory and displayed as shown below. With reference to the displayed information, the prime implicants are shown in memory.
registers from R79 down to the second level above 0 indication. In this case R77. The number in the last memory register indicates the next memory address to be interrogated for additional prime implicants, in this case starting at register R29 and decrementing until a 0 is found. All prime implicants must be recorded for future program use in the problem solution.

<table>
<thead>
<tr>
<th>Register/Contents</th>
<th>In Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>R79</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>R78</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>R77</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>R76</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

As shown above, two sets of prime implicants are listed in registers 78 and 79. In register 77, 29 points to another location in memory where the number 15 is located. A 0 follows the numbers 29 and 15, signaling the end of data in each section of memory.

The recorded possible prime implicants are: 15, 28, 35.

Step 3-A) Press key 1; read PASS 2 (side 1); press key 2; read PASS 2 (side 2); and then press key A.

B) After PASS 2 is completed, read SELECTION (side 2), and press key A.

C) Upon completion of SELECTION (side 2), the prime implicants are stored in memory and displayed as follows. With reference to the displayed information, the prime implicants are shown in memory registers from R79 down to the second level above 0 indication, in this case R77. The number in the last memory register indicates the next memory address to be interrogated for additional prime implicants, in this case starting at register R49 and decrementing until a 0 is found. All prime implicants must be recorded for future program use in the problem solution.
As shown above, two sets of prime implicants are listed in registers 78 and 79. In register 77, 49 points to another location in memory where the prime implicant (13, 15) is located in register 49 and the two prime implicants (7, 15) and (11, 15) are located in register 48. A 0 follows the numbers 49 and 7151115 signaling the end of data in each section of memory.

The recorded possible prime implicants are:

\[(3, 35), (6, 7), (7, 15), (11, 15), (13, 15).\]

D) Eliminating redundant prime implicants, the new list from PASS 1 and PASS 2 is - 28, (3, 35), (6, 7), (11, 15), (13, 15).

Step 4-A) Press Key 1; read PASS 3 (side 1); press key A.

B) After PASS 3 is completed, the prime implicants are stored in memory and displayed as follows. With reference to the displayed information, the prime implicants are shown in memory registers from R84 down to the second level above 0 indication, in this case R80. The number in the last memory register indicates the next memory address to be interrogated for additional prime implicants, in this case starting at register R32 and incrementing until a 0 is found. All prime implicants must be recorded for future program use in the problem solution.
As shown, four sets of prime implicants are listed in registers 81, 82, 83, 84. In register 80, 32 points to another location in memory where the numbers listed in registers 32 through 37 are. A 0 follows the numbers 32 and 9131115 signaling the end of data in each section of memory.

The recorded possible prime implicants are:

\[(2, 3, 6, 7), (4, 5, 6, 7), (3, 7, 11, 15), (5, 7, 13, 15), (9, 11, 13, 15), (9, 11, 13, 15).\]

C) Eliminating redundant prime implicants, the new list from all three passes is - 28, (3, 35), (2, 3, 6, 7), (4, 5, 6, 7) (9, 11, 13, 15).

D) The 3-cubes are found starting in register 14 (2 registers per cube).

These six groups form one 3-cube

\[(1, 3, 5, 7, 9, 11, 13, 15)\]
E) Recording these numbers and combining them with the other prime implicants and eliminating redundant implicants, the final list is:

(28), (3, 35), (2, 3, 6, 7), (4, 5, 6, 7),
(1, 3, 5, 7, 9, 11, 13, 15).

Step 5 - The minimal realization for the desired function has been achieved.

The run time on the calculator for this program was 19 minutes while the time to solve the same problem by hand with one repeat for check purposes took approximately 14 minutes.
IV. CONCLUSION

The objective for developing this program was to provide an aid to the digital designer. The sacrifice of program execution time for program code and memory size (i.e., routines to pack and move numbers in and out of registers is very costly in execution time), and the actual speed of the calculator resulted in the program running slower than if accomplished by hand, by a person experienced with both Quine-McCluskey and Karnaugh Maps. Because of this, when faced with either an extremely simple or a complicated problem, it would not be time or cost effective to use this program. However, in the solution of a problem of intermediate complexity, this program would be very beneficial. The time/cost for a solution would be about the same as if accomplished by hand and would free the designer to perform other tasks while the program is running. When more steps and/or memory become available, the speed of this program will be significantly increased requiring less time than if done by hand. At this point, problems in all complexities would be easily suited for this redesigned program and increase the desirability to use it.

Currently there is only one other hand-held calculator that would use this program, the HP-41C. To adapt from the TI-59 to the HP-41C would be a relatively simple task, and because of the greater amount of steps/memory available, the speed as discussed above can be achieved.

As the power of the hand-held calculators increase, so will the power of the Quine-McCluskey technique on them.
APPENDIX A

TI-59 Program Listing
## APPENDIX A

### SELECTION -

**Side 1**

<table>
<thead>
<tr>
<th>000</th>
<th>76</th>
<th>LBL</th>
<th>046</th>
<th>22</th>
<th>INV</th>
<th>092</th>
<th>07</th>
<th>7</th>
<th>011</th>
<th>69</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>11</td>
<td>A</td>
<td>047</td>
<td>67</td>
<td>EQ</td>
<td>093</td>
<td>09</td>
<td>9</td>
<td>012</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>002</td>
<td>08</td>
<td>8</td>
<td>048</td>
<td>00</td>
<td>00</td>
<td>094</td>
<td>42</td>
<td>STO</td>
<td>013</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>003</td>
<td>69</td>
<td>OP</td>
<td>049</td>
<td>55</td>
<td>55</td>
<td>095</td>
<td>09</td>
<td>09</td>
<td>014</td>
<td>72</td>
<td>ST*</td>
</tr>
<tr>
<td>004</td>
<td>17</td>
<td>17</td>
<td>050</td>
<td>00</td>
<td>0</td>
<td>096</td>
<td>04</td>
<td>4</td>
<td>015</td>
<td>06</td>
<td>06</td>
</tr>
<tr>
<td>005</td>
<td>47</td>
<td>CMS</td>
<td>051</td>
<td>42</td>
<td>STO</td>
<td>097</td>
<td>00</td>
<td>0</td>
<td>016</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>006</td>
<td>02</td>
<td>2</td>
<td>052</td>
<td>08</td>
<td>08</td>
<td>098</td>
<td>42</td>
<td>STO</td>
<td>017</td>
<td>04</td>
<td>4</td>
</tr>
<tr>
<td>007</td>
<td>06</td>
<td>6</td>
<td>053</td>
<td>69</td>
<td>OP</td>
<td>099</td>
<td>05</td>
<td>05</td>
<td>018</td>
<td>42</td>
<td>STO</td>
</tr>
<tr>
<td>008</td>
<td>42</td>
<td>STO</td>
<td>054</td>
<td>20</td>
<td>20</td>
<td>100</td>
<td>01</td>
<td>1</td>
<td>019</td>
<td>04</td>
<td>04</td>
</tr>
<tr>
<td>009</td>
<td>00</td>
<td>00</td>
<td>055</td>
<td>92</td>
<td>RTN</td>
<td>101</td>
<td>05</td>
<td>5</td>
<td>020</td>
<td>09</td>
<td>9</td>
</tr>
<tr>
<td>010</td>
<td>42</td>
<td>STO</td>
<td>056</td>
<td>76</td>
<td>LBL</td>
<td>102</td>
<td>42</td>
<td>STO</td>
<td>021</td>
<td>42</td>
<td>STO</td>
</tr>
<tr>
<td>011</td>
<td>09</td>
<td>09</td>
<td>057</td>
<td>13</td>
<td>C</td>
<td>103</td>
<td>00</td>
<td>00</td>
<td>022</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>012</td>
<td>42</td>
<td>STO</td>
<td>058</td>
<td>43</td>
<td>RCL</td>
<td>104</td>
<td>43</td>
<td>RCL</td>
<td>023</td>
<td>08</td>
<td>8</td>
</tr>
<tr>
<td>013</td>
<td>15</td>
<td>15</td>
<td>059</td>
<td>07</td>
<td>07</td>
<td>105</td>
<td>06</td>
<td>06</td>
<td>024</td>
<td>04</td>
<td>4</td>
</tr>
<tr>
<td>014</td>
<td>01</td>
<td>1</td>
<td>060</td>
<td>55</td>
<td>55</td>
<td>106</td>
<td>42</td>
<td>STO</td>
<td>025</td>
<td>42</td>
<td>STO</td>
</tr>
<tr>
<td>015</td>
<td>06</td>
<td>6</td>
<td>061</td>
<td>05</td>
<td>5</td>
<td>107</td>
<td>08</td>
<td>08</td>
<td>026</td>
<td>07</td>
<td>07</td>
</tr>
<tr>
<td>016</td>
<td>42</td>
<td>STO</td>
<td>062</td>
<td>95</td>
<td>=</td>
<td>108</td>
<td>69</td>
<td>OP</td>
<td>027</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>017</td>
<td>06</td>
<td>06</td>
<td>063</td>
<td>22</td>
<td>INV</td>
<td>109</td>
<td>38</td>
<td>38</td>
<td>028</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>018</td>
<td>02</td>
<td>2</td>
<td>064</td>
<td>59</td>
<td>INT</td>
<td>110</td>
<td>09</td>
<td>9</td>
<td>029</td>
<td>42</td>
<td>STO</td>
</tr>
<tr>
<td>019</td>
<td>06</td>
<td>6</td>
<td>065</td>
<td>32</td>
<td>X=T</td>
<td>111</td>
<td>42</td>
<td>STO</td>
<td>030</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>020</td>
<td>42</td>
<td>STO</td>
<td>066</td>
<td>00</td>
<td>0</td>
<td>112</td>
<td>06</td>
<td>06</td>
<td>031</td>
<td>02</td>
<td>2</td>
</tr>
<tr>
<td>021</td>
<td>15</td>
<td>15</td>
<td>067</td>
<td>67</td>
<td>EQ</td>
<td>113</td>
<td>91</td>
<td>R/S</td>
<td>032</td>
<td>02</td>
<td>2</td>
</tr>
<tr>
<td>022</td>
<td>01</td>
<td>1</td>
<td>068</td>
<td>00</td>
<td>0</td>
<td>114</td>
<td>76</td>
<td>LBL</td>
<td>033</td>
<td>42</td>
<td>STO</td>
</tr>
<tr>
<td>023</td>
<td>00</td>
<td>0</td>
<td>069</td>
<td>72</td>
<td>72</td>
<td>115</td>
<td>16</td>
<td>A'</td>
<td>034</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>024</td>
<td>42</td>
<td>STO</td>
<td>070</td>
<td>69</td>
<td>OP</td>
<td>116</td>
<td>86</td>
<td>STF</td>
<td>035</td>
<td>69</td>
<td>OP</td>
</tr>
<tr>
<td>025</td>
<td>05</td>
<td>05</td>
<td>071</td>
<td>20</td>
<td>20</td>
<td>117</td>
<td>03</td>
<td>03</td>
<td>036</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>026</td>
<td>92</td>
<td>RTN</td>
<td>072</td>
<td>42</td>
<td>RCL</td>
<td>118</td>
<td>91</td>
<td>R/S</td>
<td>037</td>
<td>91</td>
<td>R/S</td>
</tr>
<tr>
<td>027</td>
<td>76</td>
<td>LBL</td>
<td>073</td>
<td>00</td>
<td>0</td>
<td>119</td>
<td>76</td>
<td>LBL</td>
<td>038</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>028</td>
<td>12</td>
<td>B</td>
<td>074</td>
<td>72</td>
<td>ST*</td>
<td>120</td>
<td>17</td>
<td>B'</td>
<td>039</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>029</td>
<td>32</td>
<td>X=T</td>
<td>075</td>
<td>06</td>
<td>06</td>
<td>121</td>
<td>86</td>
<td>STF</td>
<td>040</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>030</td>
<td>01</td>
<td>1</td>
<td>076</td>
<td>43</td>
<td>RCL</td>
<td>122</td>
<td>04</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>031</td>
<td>00</td>
<td>0</td>
<td>077</td>
<td>07</td>
<td>07</td>
<td>123</td>
<td>91</td>
<td>R/S</td>
<td>PASS1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>032</td>
<td>00</td>
<td>0</td>
<td>078</td>
<td>72</td>
<td>ST*</td>
<td>000</td>
<td>76</td>
<td>LBL</td>
<td>001</td>
<td>11</td>
<td>A</td>
</tr>
<tr>
<td>033</td>
<td>64</td>
<td>PD*</td>
<td>079</td>
<td>05</td>
<td>05</td>
<td>002</td>
<td>87</td>
<td>IFF</td>
<td>003</td>
<td>03</td>
<td>03</td>
</tr>
<tr>
<td>034</td>
<td>00</td>
<td>00</td>
<td>080</td>
<td>69</td>
<td>OP</td>
<td>004</td>
<td>10</td>
<td>E'</td>
<td>005</td>
<td>22</td>
<td>INV</td>
</tr>
<tr>
<td>035</td>
<td>32</td>
<td>X=T</td>
<td>081</td>
<td>25</td>
<td>25</td>
<td>006</td>
<td>76</td>
<td>LBL</td>
<td>007</td>
<td>03</td>
<td>03</td>
</tr>
<tr>
<td>036</td>
<td>74</td>
<td>SM*</td>
<td>082</td>
<td>69</td>
<td>OP</td>
<td>008</td>
<td>43</td>
<td>RCL</td>
<td>009</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>037</td>
<td>00</td>
<td>00</td>
<td>083</td>
<td>26</td>
<td>26</td>
<td>002</td>
<td>09</td>
<td>9</td>
<td>005</td>
<td>22</td>
<td>INV</td>
</tr>
<tr>
<td>038</td>
<td>69</td>
<td>OP</td>
<td>084</td>
<td>00</td>
<td>0</td>
<td>003</td>
<td>69</td>
<td>OP</td>
<td>006</td>
<td>86</td>
<td>STF</td>
</tr>
<tr>
<td>039</td>
<td>27</td>
<td>27</td>
<td>085</td>
<td>42</td>
<td>STO</td>
<td>004</td>
<td>17</td>
<td>17</td>
<td>007</td>
<td>03</td>
<td>03</td>
</tr>
<tr>
<td>040</td>
<td>69</td>
<td>OP</td>
<td>086</td>
<td>07</td>
<td>07</td>
<td>005</td>
<td>69</td>
<td>OP</td>
<td>008</td>
<td>69</td>
<td>OP</td>
</tr>
<tr>
<td>041</td>
<td>28</td>
<td>28</td>
<td>087</td>
<td>42</td>
<td>STO</td>
<td>006</td>
<td>33</td>
<td>33</td>
<td>009</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>042</td>
<td>05</td>
<td>5</td>
<td>088</td>
<td>08</td>
<td>08</td>
<td>007</td>
<td>43</td>
<td>RCL</td>
<td>010</td>
<td>43</td>
<td>RCL</td>
</tr>
<tr>
<td>043</td>
<td>32</td>
<td>X=T</td>
<td>089</td>
<td>92</td>
<td>RTN</td>
<td>008</td>
<td>03</td>
<td>03</td>
<td>011</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>044</td>
<td>43</td>
<td>RCL</td>
<td>090</td>
<td>76</td>
<td>LBL</td>
<td>009</td>
<td>72</td>
<td>ST*</td>
<td>012</td>
<td>72</td>
<td>ST*</td>
</tr>
<tr>
<td>045</td>
<td>08</td>
<td>08</td>
<td>091</td>
<td>14</td>
<td>D</td>
<td>010</td>
<td>06</td>
<td>06</td>
<td>013</td>
<td>06</td>
<td>06</td>
</tr>
</tbody>
</table>
183 87 87
184 53 (
185 53 ( 
186 43 RCL 
187 89 89
188 75 -
189 43 RCL 
190 88 88 
191 54 )
192 65 X 
193 01 1 
194 00 0 
195 00 0 
196 54 )
197 85 + 
198 53 ( 
199 43 RCL 
200 87 87 
201 75 -
202 43 RCL 
203 88 88 
204 54 )
205 95 =
206 92 RTN
APPENDIX B

Visible Control Logic Representation (VCLR) Description
The basis for the technique of visible control logic representation (VCLR) was developed in 1973 by I. Nassi and B. Schneiderman to be compatible with structured approaches.

A visible control logic representation is a picture of a software design and its primary purpose is to enable software designers the ability to express design in a visual, structured, compact format. Concentration is on the control logic of the software design.

Standard constructs in visible control logic representations are the same as those for pseudo-code: SEQUENCE, IFTHENELSE, DOCASE, DOWHILE, DOUNITL and DOLOOP. Only the representations differ.

In the following descriptions, the word "process" is defined to consist of either a single calculation or CALL statement, or any combination of structured constructs connected in a downward manner by single entry, single exit points.

A SEQUENCE is defined as two processes executed consecutively. If "P1" and "P2" are the processes, the sequence would appear in a flow chart, VCLR or pseudo-code as shown in Figure 5.

The IFTHENELSE involves a condition test for either true or false followed by execution of one of two processes. True should be shown to the left; false to the right. In the VCLR the true path appears on the left side under the "T". In pseudo-code the true path comes immediately after the IF. If "C1" is the condition being tested, "P1" is the true path and "P2" is the false path, the
Flow Chart

'Sequence'

Figure 5.

Flow Chart

'IFTHENELSE'

Figure 6.
VCLR

Pseudo-code

'IFTHENELSE construct with NULL path'

Figure 7.

'NESTED IFTHENELSE'

Figure 8.
IFTHENELSE construct would be written as shown in Figure 6. If one of the paths is NULL, the VCLR and pseudo-code would be written as shown in Figure 7. Nested IFTHENELSEs are constructed using the standard format shown in Figure 8. The standard indentation of the three columns under the IF or under the ELSE, and the columnar alignment of the IF, ELSE, and ENDIF should be noted.

The DOCASE is shown in Figure 9. It can be implemented by the FORTRAN "computed go to" function or assembly language "jump on index" function.

DOCASE is essentially the same as IFTHENELSE except that the path selection is greater than two and all paths exit from the construct at point B.

In all cases shown, "P5" is the construct executed under the default condition. (X) is equivalent to the nested IFTHENELSE form shown in (Y). The standard indentation of the three columns under the DOCASE in (X) and IF and ELSE and ELSE IF in (Y), and the columnar alignment of DOCASE and ENDCASE: IF, ELSE IF, ELSE, and END IF should all be noted. Figure 10 shows a nested IFTHENELSE format that is equivalent to the DOCASE in Figure 9.

The DOWHILE is a loop construct with the following characteristics:

1) A counter or condition to be incremented or changed is initialized before entering the loop.

2) A test of the counter or condition is performed at the beginning of the loop.

3) The process to be executed must be a standard construct or a single statement.
Flow Chart

(X) DOCASE C1

- C1 = 1
  - P1
- C1 = 2
  - P2
- C1 = 3
  - P3
- C1 = 4
  - P4
- C1 = Other
  - P5
ENDCASE

(Y) IF C1 = 1
  - P1
ELSEIF C1 = 2
  - P2
ELSEIF C1 = 3
  - P3
ELSEIF C1 = 4
  - P4
ELSE
  - P5
ENDIF

Pseudo-code

'DOCASE'

Figure 9.
Cl = 1
C2 = 2
C3 = 3
C4 = 4

IF C1
   P1
ELSE
   IF C1
      P2
   ELSE
      IF C3
         P3
      ELSE
         IF C4
            P4
         ELSE
            P5
      ENDIF
   ENDIF
ENDIF

Pseudo-code

'Nested IFTHENELSE Equivalent to DOCASE'

Figure 10.

Flow Chart

VCLR

'DO WHILE'

Figure 11.
4) The counter or condition is incremented or modified either by the process or at the end of the loop.

5) The DOWHILE is terminated when the test counter or condition becomes false.

If the "Cl" is the condition which must exist for the loop to be executed, and "Pl" is a standard construct or a single statement, the DOWHILE would be written as shown in Figure 11.

The DOUNTIL is a loop construct with the following characteristics:

1) A counter or other condition to be incremented is initialized before entering the loop.

2) A process to be executed must be a standard construct or a single statement.

3) The counter or condition is incremented or modified either by the process or near the end of the loop.

4) The test is performed at the end of the loop.

5) The DOUNTIL is terminated when the test counter or condition becomes true.

If "Cl" is the condition which is tested after executing "Pl", and "Pl" is a standard construct or a single statement, the DOUNTIL would be written as shown if Figure 12.

The DOLOOP is a loop construct with the following characteristics:

1) The condition to be tested within the loop is initialized before entering the loop.

2) The test may be performed anywhere between the LOOP and ENDLOOP statement.

3) The condition to be met to exit from the loop is tested by the EXITIF statement.

4) The condition for exit is modified within the loop to provide the means by which exiting from the loop is accomplished.
Figure 12.

Flow Chart

VCLR

Pseudo-code

'DOUNTIL'

Figure 13.

Flow Chart

VCLR

Pseudo-code

'DOLOOP'
5) Either the process before or after the EXITIF may be null, but not both.

6) The processes to be executed must be standard constructs or single statements.

If "C1" is the condition which must be true to exit from the loop, and "P1" and "P2" are either standard constructs or single statements, the DOLOOP would be written as shown in Figure 13.

In the previous sections the major portions of the VCLR concept are discussed. With these discussions, application of these ideas in much the same way flow charts are handled in the development of software will tend to lead to a more structured way of describing the flow of data in a particular software package.
List of References


