DETERMINATION OF PRIME IMPLICANTS FOR DISJUNCTIVE BOOLEAN FUNCTIONS BY USE OF A DIGITAL COMPUTER

By

WILLIAM JOSEPH VIPRAIO
Bachelor of Science
United States Military Academy
West Point, New York
1954

Submitted to the Faculty of the Graduate School of
the Oklahoma State University
in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE
January, 1960

DETERMINATION OF PRIME IMPLICANTS FOR
DISJUNCTIVE BOOLEAN FUNCTIONS
BY USE OF A DIGITAL COMPUTER

Thesis Approved:

Paul A. McCollum
Thesis Adviser

Olen F. Cameron

Robert MacVicar
Dean of the Graduate School

PREFACE

The minimization of Boolean Functions may be broken down into two parts; the first part being the determination of the set of prime implicants, and the second being the selection from the set of prime implicants of those terms required to make up the minimal forms of the Boolean function or expression. This paper will deal with the first part, namely, the determination of the set of prime implicants. In this thesis, Boolean expressions and a computer program will be developed in order to find the set of prime implicants.

The author wishes to express his indebtedness to Frank E. McFarlin, Project Engineer for the International Business Machines Corporation, for his invaluable ideas and assistance in the preparation of this paper. Many thanks are due Professor Paul A. McCollum who acted as the writer's major advisor. Grateful acknowledgment is also due Professor William Granet for making the facilities of the Oklahoma State University Computing Center available, and also for the interest and encouragement given to the writer. Thanks are also due to the staff of the Oklahoma State University Computing Center for their generous assistance.

TABLE OF CONTENTS

CHAPTER I

THE PROBLEM

A. Statement of the Problem

In recent years, there has been an extremely rapid development of complex switching networks such as are found in modern electronic digital computers, automatic telephone dialing systems, and other complex systems so prevalent in this age of automation. For reasons of reliability, simplicity, and economy, the engineer and circuit designer has found it expedient to construct these complex switching networks of two valued or binary elements. Relays, vacuum tubes, diodes, transistors and magnetic cores are among the more common devices. The presence or absence of an electrical signal, a high or low voltage, a magnetic field of positive or negative polarity, represent some of the schemes of representing binary information. Of necessity, paralleling the development of these switching networks, an algebra of logic designed to present a mathematical expression for complicated switching operations has received much study. The algebra of logic, more commonly known as Boolean Algebra, after George Boole (1815-1864), who first introduced it in 1847 in a paper dealing with the mathematical analysis of logic, has received the attention of many authors who have since devoted much time to the problem of simplification or minimization of Boolean expressions.

W. V. Quine[1] has shown that minimization of Boolean functions may be considered in two parts, namely, the determination of a set of prime implicants, and the selection from the set of prime implicants of those terms necessary to make up minimal forms of the original Boolean expression. This paper will deal only with the first part of the problem, the determination of the set of prime implicants for a Boolean function through the use of the IBM 650 digital computer.

B. Definition of Terms

Although it is assumed that the reader has a basic understanding of Boolean Algebra, a few definitions will be given to preclude any misunderstanding on the part of the reader. Since this paper will be concerned with no more than ten independent binary variables, they will be represented by the letters A, B, C, D, E, F, G, H, I, and J.

The negative (complement, inverse) of a variable A will be written as $\overline{A}$.

A single variable, either complemented or uncomplemented, will be referred to as a literal.

The symbol + represents alternation (disjunction, inclusive OR, logical sum, inclusive union).

---

[1]W. V. Quine, "The Problem of Simplifying Truth Functions", American Mathematical Monthly, Vol. 59, October 1952, pp. 521-531.

The symbol ∘ represents conjunction (logical product, AND, intersection). The conjunction of two literals A and B will be shown as AB meaning A•B.

A term will mean a conjunction of literals.

An alterm will mean an alternation of literals.

A normal form (or disjunctive, or alternational form) will mean an alternation of terms.

A conjunctive form is a conjunction of alterms.

A term X will be said to subsume a term Y if all the literals, whether complemented or uncomplemented, whose conjunction is Y are included among the literals whose conjunction is X.

If a term X subsumes a term Y, then X implies Y.

The prime implicants of a Boolean expression will be defined as all the terms derivable from the expression such that no term or terms are subsumed by another term.

A normal canonical form for a function of n variables will mean an alternation of terms in which all n variables appear in each term.

C. Some Minimization Techniques

Many techniques for the minimization of Boolean functions have been developed, but careful investigation will reveal that most methods merely provide an alternative procedure for finding Quine's prime implicants and then selecting the necessary prime implicants to make up the Boolean function. An attempt by the author to program the Quine Method on the

IBM 650 Computer for determing prime implicants proved impractical because of the excessive number of operations required for a Boolean function of ten variables. The Harvard Computer Group have devised a chart method for the simplification of Boolean functions, but it is merely a variation of the Quine Technique. A special form of Venn diagram called the Veitch[2] diagram has been used with success in simplifying Boolean functions, however, this method is not readily adaptable for programming on a digital computer, and is even impractical for hand computation if more than a few variables are involved. Both the Quine and Harvard methods require that the Boolean expression be in the normal canonical form prior to the reduction process, while the Veitch method requires only that the expression be in normal form. Excellent concise explanations of the above three methods together with numerical examples may be found in a book by Montgomery Phister, Jr.[3]

Urbano and Mueller[4], and also Roth[5], have presented topological approaches to the minimization problem. The works of

---

[2]E. W. Veitch, "A Chart Method for Simplifying Truth-Functions," Proceedings of the Association for Computing Machinery, May 1952, pp. 128-133.

[3]Montgomery Phister, Jr., Logical Design of Digital Computers, (New York, 1958), pp. 68-108.

[4]R. H. Urbano and R. K. Mueller, A Topological Method For the Determination of the Minimal Forms of a Boolean Function, AFCRC Tech. Rept. No. TR-56-105, USAF Cambridge Research Center (Bedford, 1956).

[5]J. P. Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems I.," Transactions of the American Mathematical Society, Vol. 88, No. 2, July, 1958, pp. 301-326.

the above authors along with others may be found in the bibliography contained at the end of this paper.

CHAPTER II

DESCRIPTION OF TECHNIQUE

A. Method of Obtaining Prime Implicants

Before proceeding any further it would be advisable to state that the material to be presented in this chapter has been extracted from a paper by Frank E. McFarlin entitled "A Technique for Minimizing Boolean Functions That Does not Require a Canonical Form", dated December 31, 1958 and proposed for publication in the IRE TRANSACTIONS ON ELECTRONIC COMPUTERS. The above paper has been extracted by F. E. McFarlin from his forthcoming PhD. thesis, "Logical Design Concepts," Oklahoma State University, Stillwater, Oklahoma.

This section will show a method whereby the complete set of prime implicants for a Boolean function in disjunctive form can be obtained without first putting the function in normal canonical form. By repeated application of the following Boolean identities, the complete list of prime implicants will be obtained.

1. $A + AB = A$
2. $AB + A\overline{B} = A$
3. $AB + \overline{A}C = AB + \overline{A}C + BC$

The first two identities are those which are applied when using the Quine method from the normal canonical form, and fulfill the requirement that no term or terms are, or can be, subsumed by another term. The third identity is to insure that all the terms derivable from the Boolean function are generated. The following very simple example will serve to illustrate the application of the method.

$$\text{Given: } F = \underset{1}{\bar{B}\bar{C}} + \underset{2}{\bar{A}\bar{C}D} + \underset{3}{AB\bar{C}D} + \underset{4}{BCD}$$

Since identities 1 and 2 cannot be applied, identity 3 is used to expand the function;

Terms 1 and 2	Theorem does not apply
Terms 1 and 3	Give $A\bar{C}D$
Terms 1 and 4	Rule applies, however, term is zero
Terms 2 and 3	Give $B\bar{C}D$
Terms 2 and 4	Give $\bar{A}BD$
Terms 3 and 4	Give ABD

The new function now contains the four original terms plus the four generated terms. By application of theorems 1 and 2 the function reduces to:

$$F = \bar{B}\bar{C} + \bar{C}D + BD$$

Since further application of the three identities does not generate any new terms, the above expression thereby contains all of the prime implicants.

B. Validity of the Method

The validity and need for theorem 3 is established as follows by considering the two Boolean equations:

$$F = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C}$$

$$F = \bar{B}\bar{C} + \bar{A}B\bar{C}$$

These two equations are equivalent, and the first may be reduced by application of identities 1 and 2 (the Quine method) to yield $F = \bar{B}\bar{C} + \bar{A}\bar{C}$. Since the Quine method requires that the Boolean expression initially be in normal canonical form, the second equation cannot be reduced by application of identities 1 and 2. The term $\bar{B}\bar{C}$ does however imply the term $\bar{A}\bar{B}\bar{C}$ which, upon application of identity 2, could be used to reduce the second term of the second equation to the required prime implicant $\bar{A}\bar{C}$. Therefore, the problem is the detection of such implied terms within terms, and the utilization of such terms to obtain the desired reduction. Assume that in a Boolean expression, two of the terms imply terms that may be combined to give a third term which is not reducible by identity 1. Such a term must then either be a prime implicant, or it can be combined by use of identity 2 with another term in the original Boolean expression for eventual reduction to a prime implicant. If the two terms of the Boolean expression are considered to be X and Y, then a term T must be found which will satisfy the following two conditions:

1. $TX + TY = T$
2. $(T - X) + (T - Y) = T$

The Venn diagram shown below illustrates the concept that T-X is by definition the conjunction of T and not X (shaded area).

$$T - X = T - TX = T\bar{X}$$

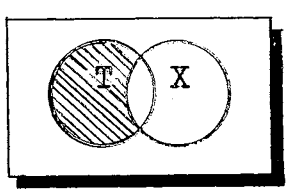


Fig. 2-1

It follows that;

$$TX = T - Y = T\overline{Y}$$

$$TY = T - X = T\overline{X}$$

Taking the second condition:

$$(T - X) + (T - Y = T$$

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

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

$$T\overline{XY} = T$$

$$T - XY = T$$

The last equation shows that if the term T is to exist, then the logical product of the two terms X and Y must be zero. This condition can only be met if, among the literals which comprise X, there is a literal which is the complement of a literal contained in Y; in other words, X = AB and Y = $\overline{A}$C, where A is a single literal and B and C are the remaining literals of X and Y. Now that the condition for the existence of the term T has been found, the term T must next be determined. Let it be assumed that it may be some function of A, $\overline{A}$, and some or all of the literals of B and C, either as contained or complemented. Rewriting the first condition as TAB + T$\overline{A}$C = T, perfect induction is next employed.

If A or $\overline{A}$ is present in T, then one of the terms on the left side of the equality will of necessity be zero and no new term can be generated. One of the terms on the left of the equality will again be zero if any complemented literal of B or C is contained in T. T must therefore be a function of B and C and must further be of the form or forms B, C, BC, or some other partial combination of the literals of B and C. If the substitution of B or C is made for T, an inequality is the result. The substitution of any partial combination of the literals of B and C also yields an inequality, therefore T = BC. If B contains a literal, and C contains the negative of the same literal, then T = 0. Thus the following Boolean identity results:

$$\bar{A}B + AC = \bar{A}B + AC + BC.$$

CHAPTER III

ANALYSIS OF PROGRAMMING TECHNIQUE

A. Boolean Types Used in Programming

Although in the previous chapter only three Boolean identities were shown for the derivation of the set of prime implicants, in programming the digital computer the following nine Boolean types were taken into consideration:

TYPE	FORM
0	$A + A = A$
1	$A + AB = A$
2	$AB + A = A$
3	$AB + A\bar{B} = A$
4	$A\bar{B} + AB = A$
5	$\bar{B}C + ABC = \bar{B}C + ABC + AC = \bar{B}C + AC$
6	$ABC + \bar{B}C = ABC + \bar{B}C + AC = \bar{B}C + AC$
7	$AC + \bar{B}\bar{C} = AC + \bar{B}\bar{C} + A\bar{B}$
8	$AC + \bar{B}D$

Type 0 is, of course, merely to eliminate duplicate terms. Type 2 is just the reverse order of type 1 and is considered so as to allow the computer to recognize the term to be eliminated. This will be brought out further on the flow chart. Types 3 and 4 are again only the reverse of each other. Types

5, 6, and 7, are all variations of the third Boolean identity presented in Chapter II. Since it is advantageous to eliminate terms whenever possible, it is desirable to allow the computer to know when the generated term is subsumed by the second term on the left of the equality (type 5), subsumed by the first term on the left of the equality (type 6), or not subsumed by either term on the left of the equality (type 7). Type 8 is a Boolean form not reducible by identities 1 and 2, nor of the form required by identity 3.

B. General Description of Program Logic

Once the terms making up the Boolean expressions have been placed as a consecutive list in computer storage, tests are then made to determine which of the nine Boolean types are present. The first term is picked up and worked against the second term, testing for the nine Boolean types in the order previously listed. After the first term has been compared against the rest of the list, the first term is stepped (the second term now being considered as the first term). The new first term is now picked up and worked against the remainder of the list. As soon as the first term becomes the last term in the list, the stepping instruction is reset and a return is made to the top of the list picking up the first and second terms. This process is repeated until types 2, 3, and 4 no longer occur and no new terms are generated. By this means, each term in the list is compared against every other term in the list, until the complete set of prime implicants has been determined.

Although the actual working of the program is slightly more involved than the above brief explanation would lead the reader to believe, it is felt that a better understanding of the method used in programming can be gained from careful scrutiny of the flow charts rather than through a word picture of the complete operation.

CHAPTER IV

IBM 650 DIGITAL COMPUTER PROGRAM

A. Program Description

The program presented in this chapter was prepared for use with the IBM 650 Electronic Digital Computer. The coding form used was IBM's Symbolic Optimal Assembly Program, Type II. Both the SOAP program and the assembled machine language instructions are shown. The program consists of 252 instructions, and including the regions reserved for data, requires 852 drum storage locations in addition to 35 locations in immediate access storage. All three indexing registers are also used.

Although the region reserved for the input data and for the storage of terms generated by the program consists of 301 locations, it is strongly advised that no more than 50 terms of a Boolean function be read into storage at one time. This will allow sufficient room for storage of generated terms and will also decrease the computation time. For Boolean expressions of exceptional length, it desirable to break down the function into blocks of ten to twenty terms per block and to find the prime implicants for each block as if they were individual expressions; the results may then be combined to yield the final set of prime implicants by feeding the reduced data to the computer. Should this advice be disregarded, and too large an amount of data be fed to the computer so that the program attempts to store a generated term outside of the reserved region, a built-in stop code will cause the computer to halt operations without punching out any cards. Should this occur, the advice given above should be heeded, and the loading started anew.

The program is designed to handle Boolean expressions in disjunctive form, each term consisting of no more than ten literals.

It is also recommended that prior to initially loading the program on the drum, a core and drum clearing routine be used to clear all storage locations. Pre-punched clearing routines can usually be found in any computing center, or should this not be the case, the IBM 650 Operating Manual contains a satisfactory clearing procedure. Once the program has been loaded, no further clearing is necessary, and only the data along with the required transfer cards are needed to solve successive Boolean expressions for the set of prime implicants.

Although the program has been extensively tested, the author makes no guarantee and assumes no responsibility against the possibility of failure for a specific problem.

B. Input Requirements

Region A, consisting of drum storage locations 0000 to 0300 inclusive, has been reserved for the input data. The

Boolean terms for a specific problem should be loaded consecutively in this region commencing with location 0000. The method of loading data is left to the discretion of the reader. One-word load cards have proven very adequate for most cases, however, the reader may prefer to load the data seven-per-card, or in some other form for a Boolean expression of great length. In addition to the Boolean terms, one must also load the number of terms minus one as a problem constant into location 9000. This is important as it sets the length of the list of terms. For a Boolean expression consisting of twenty terms, the number N-1 = 20-1 = 19, and written as 0000000019, must be loaded into core storage location 9000.

Each Boolean term is ten digits in length regardless if the number of independent binary variables is less than ten. An uncomplemented literal will be represented by the numeric 1, a complemented literal by the numeric 3, and the absence of a literal by a 0. A few examples are shown below:

Boolean Term		Numeric Representation
$A \bar{B} \bar{C} D \bar{E} \bar{F} \bar{G} H I J$	=	1331333111
$\bar{A} C \bar{D} F$	=	3013010000
$G \bar{I} J$	=	0000001031

The order of input for a problem, assuming that the computer program is on five-per card format and the data is to be loaded on one-per card, is as follows:

1. Core and drum clear cards.
2. Computer program (five-per card).
3. Transfer card (L-5 to L-1).

4. N-1 card (one-per card).

5. Data cards (one-per card).

6. Transfer card (to location 0350 = start of program).

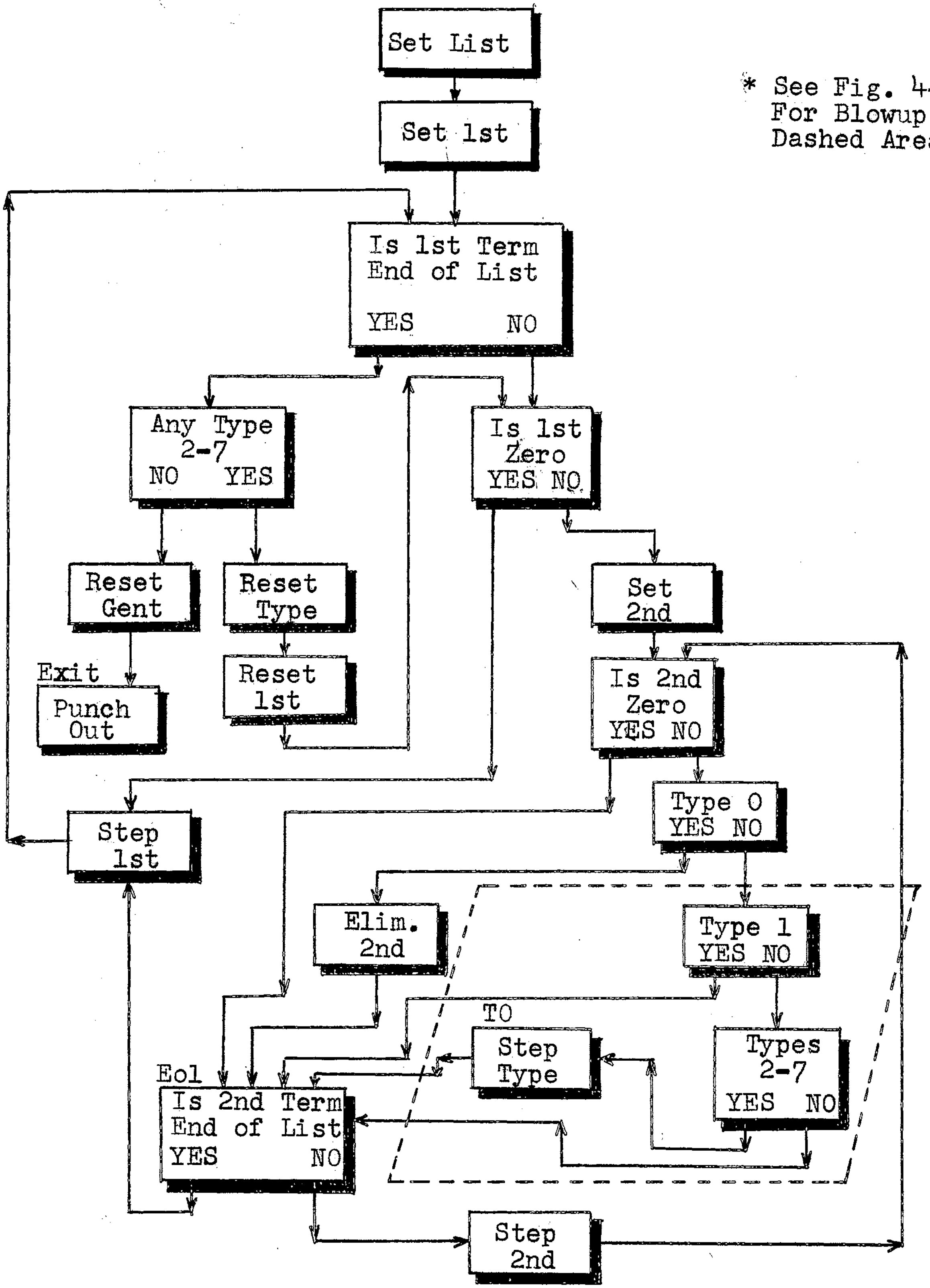
For the successive determination of the prime implicants for other Boolean expressions, only steps 3, 4, 5, and 6 are necessary.

C. Output Card Format

The program in its present form is designed to punch out the prime implicants for a disjunctive Boolean expression on a one-per card format; the first ten digits of each card being the various prime implicants. The solution is not converted to alphabetic form, but remains coded in the numeric form as discussed under input requirements.

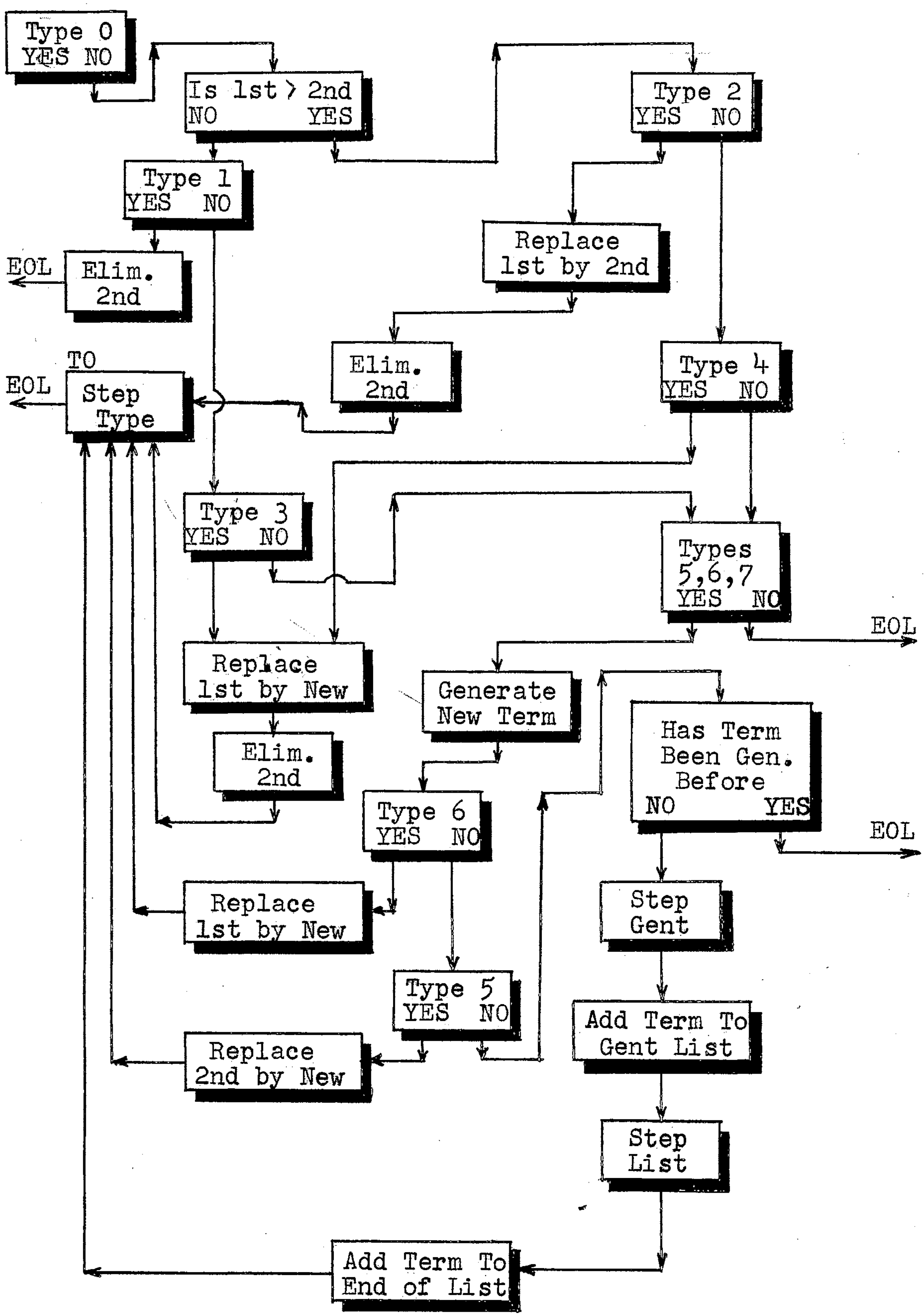
D. Flow Chart

The flow charts presented in Figures 4-1 and 4-2 represent the actual technique used in the application of the tests for the nine Boolean types. A study of these charts will give the reader a good understanding of the program logic. Should the reader desire to modify the program in any way, these charts will aid greatly.



* See Fig. 4-2 For Blowup of Dashed Area

Fig. 4-1



Type 0
YES NO
Is 1st > 2nd
NO YES
Type 2
YES NO
Type 1
YES NO
Replace
1st by 2nd
EOL
Elim.
2nd
Elim.
2nd
Type 4
YES NO
TO
EOL
Step
Type
Type 3
YES NO
Types
5,6,7
YES NO
EOL
Replace
1st by New
Generate
New Term
Has Term
Been Gen.
Before
NO YES
Elim.
2nd
Type 6
YES NO
EOL
Replace
1st by New
Step
Gent
Type 5
YES NO
Add Term To
Gent List
Replace
2nd by New
Step
List
Add Term To
End of List

Fig. 4-2

E. Computer Program

The program for the IBM 650 Digital Computer, which was mentioned in the preceeding pages is shown below and on the following pages. The compiled machine language instructions are shown on the left while the corresponding SOAP II instructions are shown to the right.

	MACHINE LANGUAGE				SOAP II				
Inst. No.	Location	Op. Code	Data Add.	Inst. Add.	Location	Op. Code	Data Add.	Tag	Inst. Add.
0001	0000	00	0000	0000		REG	A0000		0300
0002	0000	00	0000	0000		REG	B1700		1999
0003	0000	00	0000	0000		EQU	GENT		9034
0004	0000	00	0000	0000		EQU	FORM		9032
0005	0000	00	0000	0000		EQU	LIST		9000
0006	0000	00	0000	0000		EQU	TYPE		9001
0007	0350	80	0000	0306	START	RAA	0000		1ST
0008	0306	60	2000	0305	1ST	RAU	A0001	A	
0009	0305	69	9000	0312		LDD	LIST		
0010	0312	51	8001	0318		SXA	8001		
0011	0318	40	0321	0322		NZA	N1		
0012	0322	60	9001	0331		RAU	TYPE		
0013	0331	44	0335	0336		NZU			EXIT
0014	0335	11	8003	0345		SUP	8003		
0015	0345	21	9001	0304		STU	TYPE		
0016	0304	80	0000	0306		RAA	0000		1ST
0017	0321	50	8001	0327	N1	AXA	8001		

0018	0327	44	0381	0332		NZU	N2		
0019	0332	50	0001	0306		AXA	0001		1ST
0020	0381	69	8005	0338	N2	LDD	8005		
0021	0338	82	8001	0344		RAB	8001		
0022	0344	52	0001	0400		AXB	0001		2ND
0023	0400	60	4000	0355	2ND	RAU	A0001	B	
0024	0355	44	0309	0310		NZU	N3		EOL
0025	0310	69	9000	0317	EOL	LDD	LIST		
0026	0317	53	8001	0323		SXB	8001		
0027	0323	42	0326	0377		NZB			Y1
0028	0326	52	8001	0333		AXB	8001		
0029	0333	52	0001	0400		AXB	0001		2ND
0030	0377	50	0001	0306	Y1	AXA	0001		1ST
0031	0309	11	2000	0405	N3	SUP	A0001	A	
0032	0405	44	0359	0360		NZU	N4		
0033	0360	21	4000	0310		STU	A0001	B	EOL
0034	0359	69	0362	0315	N4	LDD	VAR		
0035	0315	89	8001	0371		RSC	8001		
0036	0371	65	2000	0455		RAL	A0001	A	N5
0037	0455	35	0001	0311	N5	SLT	0001		
0038	0311	21	9612	0320		STU	9012	C	
0039	0320	11	8003	0329		SUP	8003		
0040	0329	58	0001	0385		AXC	0001		
0041	0385	48	0455	0339		NZC	N5		
0042	0339	69	0362	0365		LDD	VAR		
0043	0365	89	8001	0421		RSC	8001		
0044	0421	65	4000	0505		RAL	A0001	B	N6

0045	0505	35	0001	0361	N6	SLT	0001		
0046	0361	21	9622	0370		STU	9022	C	
0047	0370	11	8003	0379		SUP	8003		
0048	0379	58	0001	0435		AXC	0001		
0049	0435	48	0505	0389		NZC	N6		
0050	0389	69	0362	0415		LDD	VAR		
0051	0415	89	8001	0471		RSC	8001		
0052	0471	60	2000	0555		RAU	A0001	A	
0053	0555	11	4000	0605		SUP	A0001	B	
0054	0605	46	0308	0409		BMI	T1		T2
0055	0308	60	9612	0367	T1	RAU	9012	C	
0056	0367	44	0521	0372		NZU			Y2
0057	0521	11	9622	0431		SUP	9022	C	
0058	0431	44	0485	0372		NZU	T3		Y2
0059	0372	58	0001	0328	Y2	AXC	0001		
0060	0328	48	0308	0382		NZC	T1		
0061	0382	21	4000	0310		STU	A0001	B	EOL
0062	0409	60	9622	0417	T2	RAU	9022	C	
0063	0417	44	0571	0422		NZU			Y3
0064	0571	11	9612	0481		SUP	9012	C	
0065	0481	44	0535	0422		NZU	T4		Y3
0066	0422	58	0001	0378	Y3	AXC	0001		
0067	0378	48	0409	0432		NZC	T2		
0068	0432	60	4000	0655		RAU	A0001	B	
0069	0655	21	2000	0303		STU	A0001	A	
0070	0303	11	8003	0411		SUP	8003		
0071	0411	21	4000	0353		STU	A0001	B	TD

0072	0485	60	9032	0343	T3	RAU	FORM		
0073	0343	11	8003	0301		SUP	8003		
0074	0301	21	9032	0410		STU	FORM		
0075	0410	69	0362	0465		LDD	VAR		
0076	0465	89	8001	0621		RSC	8001		3T
0077	0621	60	9622	0429	3T	RAU	9022	C	
0078	0429	11	9612	0337		SUP	9012	C	
0079	0337	46	0340	0341		BMI	T5		
0080	0341	44	0395	0346		NZU	Y6		
0081	0346	24	9632	0403		STD	9032	C	
0082	0403	58	0001	0459		AXC	0001		
0083	0459	48	0621	0313		NZC	3T		Y7
0084	0395	11	0348	0453	Y6	SUP	TWO		
0085	0453	44	0340	0358		NZU	T5		
0086	0358	21	9632	0316		STU	9032	C	
0087	0316	60	9032	0325		RAU	FORM		
0088	0325	10	0428	0383		AUP	ONE		
0089	0383	21	9032	0342		STU	FORM		
0090	0342	58	0001	0398		AXC	0001		
0091	0398	48	0621	0313		NZC	3T		Y7
0092	0313	60	9032	0671	Y7	RAU	FORM		
0093	0671	44	0375	0340		NZU			T5
0094	0375	11	0428	0433		SUP	ONE		
0095	0433	44	0340	0388		NZU	T5		
0096	0388	69	0362	0515		LDD	VAR		
0097	0515	89	8001	0721		RSC	8001		
0098	0721	21	4000	0503		STU	A0001	B	Y8

0099	0503	10	9632	0461	Y8	AUP	9032	C	
0100	0461	58	0001	0467		AXC	0001		
0101	0467	48	0420	0771		NZC			Y9
0102	0420	35	0001	0503		SLT	0001		Y8
0103	0771	21	2000	0353	Y9	STU	A0001	A	T0
0104	0535	60	9032	0393	T4	RAU	FORM		
0105	0393	11	8003	0351		SUP	8003		
0106	0351	21	9032	0460		STU	FORM		
0107	0460	69	0362	0565		LDD	VAR		
0108	0565	89	8001	0821		RSC	8001		4T
0109	0821	60	9612	0479	4T	RAU	9012	C	
0110	0479	11	9622	0387		SUP	9022	C	
0111	0387	46	0340	0391		BMI	T5		
0112	0391	44	0445	0396		NZU	Z1		
0113	0396	24	9632	0553		STD	9032	C	
0114	0553	58	0001	0509		AXC	0001		
0115	0509	48	0821	0313		NZC	4T		Y7
0116	0445	11	0348	0603	Z1	SUP	TWO		
0117	0603	44	0340	0408		NZU	T5		
0118	0408	21	9632	0366		STU	9032	C	
0119	0366	60	9032	0425		RAU	FORM		
0120	0425	10	0428	0483		AUP	ONE		
0121	0483	21	9032	0392		STU	FORM		
0122	0392	58	0001	0448		AXC	0001		
0123	0448	48	0821	0313		NZC	4T		Y7
0124	0340	60	9032	0349	T5	RAU	FORM		
0125	0349	11	8003	0307		SUP	8003		

0126	0307	21	9032	0416		STU	FORM		
0127	0416	69	0362	0615		LDD	VAR		
0128	0615	89	8001	0871		RSC	8001		
0129	0871	65	2000	0705		RAL	A0001	A	
0130	0705	15	4000	0755		ALO	A0001	B	X1
0131	0755	35	0001	0511	X1	SLT	0001		
0132	0511	44	0665	0466		NZU	X2		
0133	0466	21	9632	0324		STU	9032	C	
0134	0324	58	0001	0330		AXC	0001		
0135	0330	48	0755	0334		NZC	X1		OUT
0136	0665	11	0428	0533	X2	SUP	ONE		
0137	0533	44	0437	0438		NZU	X3		
0138	0438	24	9632	0495		STD	9032	C	
0139	0495	58	0001	0401		AXC	0001		
0140	0401	48	0755	0334		NZC	X1		OUT
0141	0437	11	0428	0583	X3	SUP	ONE		
0142	0583	44	0487	0488		NZU	X4		
0143	0488	24	9632	0545		STD	9032	C	
0144	0545	58	0001	0451		AXC	0001		
0145	0451	48	0755	0334		NZC	X1		OUT
0146	0487	11	0428	0633	X4	SUP	ONE		
0147	0633	44	0537	0538		NZU	X5		
0148	0538	69	0441	0394		LDD	TREY		
0149	0394	24	9632	0501		STD	9032	C	
0150	0501	58	0001	0357		AXC	0001		
0151	0357	48	0755	0334		NZC	X1		OUT
0152	0537	11	0428	0683	X5	SUP	ONE		

0153	0683	44	0587	0588		NZU	X6		
0154	0588	21	9632	0446		STU	9032	C	
0155	0446	10	9032	0805		AUP	FORM		
0156	0805	10	0428	0733		AUP	ONE		
0157	0733	21	9032	0442		STU	FORM		
0158	0442	11	8003	0551		SUP	8003		
0159	0551	58	0001	0407		AXC	0001		
0160	0407	48	0755	0334		NZC	X1		OUT
0161	0587	10	0428	0783	X6	AUP	ONE		
0162	0783	21	9632	0492		STU	9032	C	
0163	0492	11	8003	0601		SUP	8003		
0164	0601	58	0001	0457		AXC	0001		
0165	0457	48	0755	0334		NZC	X1		OUT
0166	0334	60	9032	0443	OUT	RAU	FORM		
0167	0443	44	0347	0310		NZU			EOL
0168	0347	11	0428	0833		SUP	ONE		
0169	0833	44	0310	0638		NZU	EOL		
0170	0638	69	0362	0715		LDD	VAR		
0171	0715	89	8001	0921		RSC	8001		R1
0172	0921	60	9632	0529	R1	RAU	9032	C	
0173	0529	44	0883	0384		NZU			R2
0174	0883	11	9612	0493		SUP	9012	C	
0175	0493	44	0397	0384		NZU	R5		R2
0176	0384	58	0001	0390	R2	AXC	0001		
0177	0390	48	0921	0444		NZC	R1		
0178	0444	69	0362	0765		LDD	VAR		
0179	0765	89	8001	0971		RSC	8001		R3

0180	0971	10	9632	0579	R3	AUP	9032	C	
0181	0579	58	0001	0585		AXC	0001		
0182	0585	48	0688	0439		NZC			R4
0183	0688	35	0001	0971		SLT	0001		R3
0184	0439	21	2000	0353	R4	STU	A0001	A	TO
0185	0397	69	0362	0815	R5	LDD	VAR		
0186	0815	89	8001	1021		RSC	8001		R6
0187	1021	60	9632	0629	R6	RAU	9032	C	
0188	0629	44	0933	0434		NZU			R7
0189	0933	11	9622	0543		SUP	9022	C	
0190	0543	44	0447	0434		NZU	SO		R7
0191	0434	58	0001	0440	R7	AXC	0001		
0192	0440	48	1021	0494		NZC	R6		
0193	0494	69	0362	0865		LDD	VAR		
0194	0865	89	8001	1071		RSC	8001		R8
0195	1071	10	9632	0679	R8	AUP	9032	C	
0196	0679	58	0001	0635		AXC	0001		
0197	0635	48	0738	0489		NZC			R9
0198	0738	35	0001	1071		SLT	0001		R8
0199	0489	21	4000	0353	R9	STU	A0001	B	TO
0200	0447	69	0362	0915	SO	LDD	VAR		
0201	0915	89	8001	1121		RSC	8001		
0202	1121	16	8002	0729		SLO	8002		
0203	0729	11	8003	0637		SUP	8003		S1
0204	0637	10	9632	0595	S1	AUP	9032	C	
0205	0595	58	0001	0651		AXC	0001		
0206	0651	48	0354	0855		NZC			S2

0207	0354	35	0001	0637		SLT	0001		S1
0208	0855	21	9033	0314	S2	STU	9033		
0209	0314	60	9034	0373		RAU	GENT		
0210	0373	44	0427	0478		NZU	S5		S7
0211	0478	10	0428	0983	S7	AUP	ONE		
0212	0983	21	9034	0542		STU	GENT		
0213	0542	88	8001	0498		RAC	8001		
0214	0498	60	9033	0507		RAU	9033		
0215	0507	21	7699	0302		STU	B0000	C	S4
0216	0427	88	8001	1033	S5	RAC	8001		S6
0217	1033	60	7699	0653	S6	RAU	B0000	C	
0218	0653	11	9033	0561		SUP	9033		
0219	0561	44	0965	0310		NZU			EOL
0220	0965	59	0001	0516		SXC	0001		
0221	0516	48	1033	0470		NZC	S6		
0222	0470	60	9034	0478		RAU	GENT		S7
0223	0302	65	9000	0611	S4	RAL	LIST		
0224	0611	15	0428	1083		ALO	ONE		
0225	1083	20	9000	0491		STL	LIST		
0226	0491	88	8001	0497		RAC	8001		
0227	0497	59	0300	0703		SXC	0300		
0228	0703	48	0356	0557		NZC			FAULT
0229	0356	58	0300	0363		AXC	0300		
0230	0363	60	9033	1171		RAU	9033		
0231	1171	21	6000	0353		STU	A0001	C	TO
0232	0557	01	0000	0000	FAULT	HLT	0000		0000
0233	0353	60	9001	0661	TO	RAU	TYPE		

0234	0661	10	0428	1133		AUP	ONE		
0235	1133	21	9001	0310		STU	TYPE		EOL
0236	0336	60	9034	0645	EXIT	RAU	GENT		
0237	0645	11	8003	0753		SUP	8003		
0238	0753	21	9034	0412		STU	GENT		
0239	0412	69	9000	0319		LDD	LIST		
0240	0319	80	8001	0475		RAA	8001		L1
0241	0475	60	2000	0905	L1	RAU	A0001	A	
0242	0905	44	0559	0510		NZU			L2
0243	0559	21	9059	0368		STU	9059		
0244	0368	71	9059	0510		PCH	9059		L2
0245	0510	40	0413	0364	L2	NZA			L3
0246	0413	51	0001	0475		SXA	0001		L1
0247	0364	01	0000	0000	L3	HLT	0000		0000
0248	0450	21	0000	0353	TERM	STU	0000		TO
0249	0362	00	0000	0010	VAR				10
0250	0428	00	0000	0001	ONE				1
0251	0348	00	0000	0002	TWO				2
0252	0441	00	0000	0003	TREY				3

CHAPTER V

SUMMARY AND CONCLUSIONS

The result of this study has been the development of a program for the IBM 650 Digital Computer which will determine the set of Prime Implicants for disjunctive Boolean functions. Every attempt has been made to hold the number of instructions to a minimum. The program was compiled using SOAP II in order to reduce computation time. The program will effectively handle Boolean expressions containing a maximum of ten variables. No restriction is made upon the number of terms comprising the Boolean function. In order to handle the Boolean expression, the program requires that the Boolean terms be written in a simple coded numeric form. The program output is in the same coded form. Another program requirement is that the Boolean expression be in the normal or disjunctive form; the normal canonical form is not necessary but is, of course, acceptable.

Although the method used in programming is quite readily adaptable to a decimal coded computer such as the IBM 650, the method is even more suitable for a binary computer such as the IBM 704. The IBM 704, in addition to being approximately thirty times faster than the IBM 650, possesses certain intrinsic qualities or more powerful operation codes, that would enable it to handle Boolean functions more effectively. Operation codes which will perform logical AND, OR, and EXCLUSIVE OR operations are available on the IBM 704 Computer. The IBM 704 can also handle words of greater and variable length, has greater storage capacity, and finally, has a masking facility which would greatly enhance the comparison of Boolean terms.

Since an IBM 704 Computer was not available for use by the author, and since there are more IBM 650 Computers in use than any other computer of comparable size, type, and speed, it was felt that a program for the IBM 650, such as presented in this paper, was a worthwhile endeavor.

An application of the Petrick Method[6] to a digital computer, utilizing the results obtained from the program presented in this paper, would yield a composite program which would find the minimal form or forms, as the case may be, for disjunctive Boolean functions.

[6] S. R. Petrick, _A Direct Determination of the Irredundant Forms of a Boolean Function From the Set of Prime Implicants_, AFCRC-TR-56-110, USAF Cambridge Research Center (Bedford, 1956).

BIBLIOGRAPHY

Ghazala, M. J. "Irredundant Disjunctive and Conjunctive Forms of a Boolean Function." I.B.M. Journal of Research and Development, I, No. 2, (April, 1957), 171-176.

Karnough, M. "The Map Method of Synthesis of Combinational Logic Circuits." A.I.E.E. Transactions, LXXII, Part 1 (November, 1953), 593-599.

McCluskey, E. J., Jr. "Minimization of Boolean Functions." The Bell System Technical Journal, XXXV (November, 1956), 1417-1444.

McFarlin, Frank E., "A Technique for Minimizing Boolean Functions That Does Not Require a Canonical Form." (pamphlet proposed for publication in the I.R.E. Transactions on Electronic Computers, Endicott, New York, December 1958).

Petrick, S. R., A Direct Determination of the Irredundant Forms of a Boolean Function from the Set of Prime Implicants. USAF Cambridge Research Center TR-56-110 (April, 1956).

Phister, Montgomery, Jr., Logical Design of Digital Computers, New York: John Wiley and Sons, Inc., 1958.

Quine, W. V., "The Problem of Simplifying Truth-Functions." The American Mathematical Monthly, LIX (October, 1952), 521-531.

________. "A Way to Simplify Truth Functions." The American Mathematical Monthly, LXII (November, 1955), 627-631.

Roth, J. P., "Algebraic Topological Methods for the Synthesis of Switching Systems I." Transactions of the American Mathematical Society, LXXXVIII, No. 2 (July, 1958), 301-326.

Samson, E. W., and R. Mueller, Circuit Minimization: Minimal and Irredundant Boolean Sums by Alternative Set Method. USAF Cambridge Research Center TR-55-109 (June, 1955).

Urbano, R. H., and Mueller, R. K., A Topological Method for the Determination of the Minimal Forms of a Boolean Function. USAF Cambridge Research Center TR-56-105 (March, 1956).

VITA

William Joseph Vipraio

Candidate for the Degree of

Master of Science

Title: DETERMINATION OF PRIME IMPLICANTS FOR DISJUNCTIVE BOOLEAN FUNCTIONS BY USE OF A DIGITAL COMPUTER

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Chicago, Illinois, September 12, 1931, the son of William and Mary Vipraio.

Education: Attended grade school in Chicago, Illinois; graduated from Bowen High School, Chicago, Illinois, in 1949; attended the University of Illinois, Chicago, Undergraduate Division, for one year, 1949 to 1950; received the Bachelor of Science Degree from the United States Military Academy in June, 1954; completed requirements for the Master of Science Degree in January, 1960. Member of Sigma Tau, I. R. E.

Professional Experience: Entered the United States Air Force in June, 1954, and after receiving pilots wings was an Aircraft Commander with the Atlantic Division of the Military Air Transport Service until attending the Oklahoma State University.