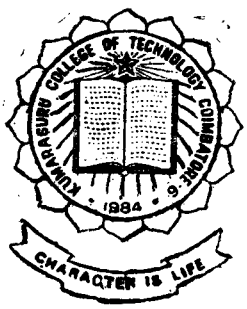
Software Package for the Design of Digital Systems

P-165

Project Report



1991-92

Submitted by

P. C. Premi

Sharmila .L

Kavitha .S

Under the Guidance of

Prof. P. Shanmugam, B.E.,M.Sc.,(Engg.),M.S. (Hawaii)

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING OF BHARATHIAR UNIVERSITY COIMBATORE

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore-641 006

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore-641 006

P-165

Certificate

This is to Certify that the report entitled

Software Package for the Design of Digital Systems

hes been Submitted by

Ms......

in partial fulfilment for the award of Bachelor of Engineering in the Computer Science and Engineering branch of the Bharathiar University, Coimbatore-641 046 during the academic year 1991-92.

.....
Guide

.....
Head of the Department

Certified that the candidate was Examined by us in the Project Work Viva - Voce Examination held on.....*and the University Register Number was*.....

.....
Internal Examiner

.....
External Examiner

ACKNOWLEDGEMENT

We wish to express our gratitude to our project guide **Prof. P. Shanmugam, B.E., M.Sc., (Engg), M.S,** Head, Department of Computer Science, for the confidence he had reposed on us, by assigning us this project, for his valuable guidance and encouragement throughout this project work.

We are thankful to the college management and to our Principal **Major T.S. Ramamurthy, B.E.,** for their kind patronage.

We also wish to thank the faculty, Department of Computer Science and Engineering for their under ending cooperation and for motivating us all through.

We are grateful to **Miss. R. Pavai, B.E.,** and **Mr. M. Gopalan, B.E.,** for their advice and encouragement during the project work.

Finally, we wish to thank all the Non-Teaching staff of the Computer Department for availing us of the LAB facilities during our project work.

SYNOPSIS

The Karnaugh map is one of the most powerful tools in the repertoire of the logic designer, but it has several drawbacks. It is a trail-and-error method. It depends on the human ability to recognize patterns which makes it unsuitable for a digital computer. In practice we often encounter logic design problems that involve variables considerably more than four; in these situations karnaugh map will be difficult to apply.

To overcome the above drawbacks, there is an algorithm called the **Quine-McCluskey algorithm.** This method involves simple, repetitive operations that compare each minterm to all other minterms with which it may form a combinable grouping. The reduced implicants that results from this process are compared among themselves to identify further reductions until no further reductions are possible. This process identifies all the prime implicants that may be used to represent the function in simplified sum-of-product

form. A selection process is then required to determine which set of prime implicants does the best job of covering the function.

The primary field of interest in our project is to write the software for obtaining the reduced equation by Quine-McCluskey method, which also draws the logical circuit. The software routines for the various processing techniques are implemented in "C" and the results are given.

CONTENTS

CHAPTER I

INTRODUCTION

1.1 The Design Of Digital Systems:

Digital design may be separated into three general aspects : component design, functional design, and systems design. Component design centers around the electronic devices that perform the simplified logical operations within a digital system. These devices have names such as gates, flip-flops etc., and their design and use is somewhat different from other aspects of electronics or circuit analysis. Functional design is the assembly of digital components into working subsystems such as counters or adders. These functional subsystems are then combined along with appropriate timing, control and interface systems, into the overall digital system that perform the desired operations. The last task is the systems design phase. Typical digital systems include traffic light controllers, numerically controlled milling machines and digital computers.

1.2 Logic Circuits-Combinational and Sequential:

Logic circuits for digital systems may be combinational or sequential. A combinational circuit can be defined from the behavioural point of view as a circuit whose output is dependent only on the inputs at the same time instant,or can be defined from the constructional point of

view as a circuit that does not contain any memory element. Sequential circuit employ memory elements in addition to logic gates. A sequential circuit can be defined from the behavioural point of view as a circuit whose output depends not only on the present inputs, but also on the past history of the inputs, or can be defined from the constructional point of view as a circuit that contains atleast one memory element.

1.2.1 Combinational Circuits:

A combinational circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the inputs and geenerate signals to the outputs. The process transforms binary information from the given input data to the required output data. Both input and output data are represented by binary signals, i.e., they exists in two possible values, one representing logic-1 and the other logic-0.

For n input variables, there are 2^n possible combinations of binary input values. For each possible input combination, there is only one possible output combination. A combinational circuit can be described by m Boolean functions, one for each output variable.Each output function is expressed in terms of the n input variables.

1.2.2 Design Procedure:

The design of the combinational circuits starts

from the verbal outline of the problem and ends in a logic circuit diagram, or a set of Boolean functions from which the logic diagram can be easily obtained.

In designing combinational digital systems, all input and output signals to and from the system are represented in True/False form and represented by Boolean variables. Algebraic expressions defining the output variables in terms of the input variables are then defined according to the specifications of the digital system. These expressions are manipulated in order to simpplify the amount of logical processing that is required to implement the system. The system is then constructed with suitable logical devices, such as electronic gating circuits, in accordance with the simplified logical equations that describe the system. The procedure involves the following steps.

1. The problem is stated.
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationships between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

1.3 Design Tools:

The Karnaugh map is a very powerful design tool. The power of Karnaugh map does not lie in its application of any marvellous new theorems but instead in its utilization of the remarkable ability of the human mind to perceive patterns in pictorial representations of data.

A Karnaugh map may be regarded either as a pictorial form of a truth table or as an eextension of the Venn diagram. Though it is a powerful design tool, it has certain drawbacks. The drawbacks are stated below.

1. It is a trial-and-error method that does not offer any guarantee of producing the best realization.
2. Its dependence on the human ability to recognize patterns, makes it unsuitable for any form of mechanization, such as programming for a digital computer.
3. For functions of four or more variables, it is difficult for the designer to be sure that he has selected the smallest possible set of products from a karnaugh map.

To overcome the drawbacks of the karnaugh map there is an algorithm called Quine-McCluskey algorithm. The basic Quine-McCluskey minimization procedure is as follows.

1. Find the prime implicants of the function.
2. Construct the prime-implicant table and find the essential prime-implicants(essential rows) of the function.
3. Include the essential prime-implicants in the minimal sum.

4. After all essential prime-implicants are deleted from the prime-implicant table, determine the dominated rows and dominated columns in the table, delete all dominating rows and dominating columns, and find the secondary(essential) prime implicants.
5. Repeat 3 and 4 as many times as they are applicable until a minimal cover of the function is found.

CHAPTER II

DESIGN TOOLS

The objective of this chapter is to give a detailed description of the Karnaugh map and the Quine-McCluskey method.

2.1 Karnaugh Map

A Karnaugh Map, hereafter called a K-Map is a very powerful design tool. It is a modification of the Venn diagram. This is a graphical method for representing a Boolean function. It is similar to a truth table, in that the map supplies the TRUE or FALSE value of a Boolean function for all possible combinations of its logical arguments. There are many ways in which a K-map can be arranged. The most important considerations of the arrangement are

1. There must be a unique location on the K-map for entering the TRUE/FALSE value of the function that corresponds to each combination of input variables.
2. The location should be arranged so that reductions of the form are readily apparent to the trained observer.

The second consideration implies that a successful K-mapping arrangement should point to groups of minterms or maxterms that can be combined into reduced forms. K-maps are also useful in expanding partially reduced expressions into standard form prior to the minimization process.

K-maps can be used to represent functions of any number of variables, but they are most useful with functions having fewer than six arguements. Higher ordered K-maps can be obtained by successfully doubling the size of the single variable K-map. One requirement for a K-map is that there must be a square corresponding to each input combination, hence there must be a 2•n square's in a K-map for n-variables.Another requirement is that squares must be so arranged that any pair of squares immediately adjacent to each other (horizontally or vertically) must correspond to a pair of input conditions that are logically adjacent, i.e., differ in only one variable.

Consider a function having three variables. For the three given inputs, the number of combinations are 2•3, i.e., 8 combinations. These eight combinations are represented in a K-map and its corresponding truth tables for the three inputs are shown in figure 2.1.

A	B	C	A . B . C
0	0	0	a b c
0	0	1	a b C
0	1	0	a B c
0	1	1	a B C
1	0	0	A b c
1	0	1	A b C
1	1	0	A B c
1	1	1	A B C

(a)

C \ AB	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

(b)

fig 2.1

In fig2.1(b), squares 3 and 7 on the maps correspond to input combinations aBC and ABC, identical except in A. Note that squares at the end of columns or rows are also logically adjacent. In fig 2.1(b), m0, m1, m2 etc correspond to minterms, i.e., for a three input function, we have eight minterms. A minterm is the form of Boolean function corresponding to the minimum possible area, other than 0, on a K-map.

2.2 Simplification Of Karnaugh Maps:

To simplify a given function initially mark those squares with a 1 in the map for which the minterms are given. After entering 1s on the K-mao, enncircle the octet first, the quads second and the pairs last.

PAIRS: Two horizontally or vertically adjacent 1s on a K-map is a pair. A pair eliminates a single variable and its complement.

QUAD: A quad is a group of four 1s that are horizontally or vertically adjacent. The 1s may be end to end or in the form of a square. A quad leads to a simpler product than a pair, i.e., a quad eliminates two variables and their complements.

OCTET: The octet is a group of eight 1s. An Octet eliminates three variables and their complements.

2.2.1 An Example:

For the given function

$$f(A,B,C,D)= \Sigma\ m(5,4,13,2,6,11,12)$$

the simplified equation is obtained using the K-map which is shown below in figure 2.2.

AB / CD	00	01	11	10
00	0	0	0	1
01	1	1	0	1
11	1	1	0	0
10	0	0	(1)	0

fig 2.2

$$f(A,B,C,D)= \Sigma\ m(5,4,13,2,6,11,12)$$

$$= aD + Abc + ABCd$$

Eliminating Redundant Groups: After encircling groups, eliminate any redundant group. Redundant group is a group of 1s on a K-map that are all part of other groups.

Don't-Care Conditions: In some digital systems, certain input conditons never occur during normal operation; Therefore, the corresponding output never appears, it is indicated by an X in the truth-table. The X is called a don't care condition. Whenever you see an X in a truth-table, you can let it be equal either to 0 or 1, whichever produces a simpler logic circuit.

2.2.2 Conclusion Of Karnaugh-Maps:

Here is the summary of the K-map method for simplifying Boolean equations:

1. Enter a 1 on the K-map for each fundamental product that produces a 1 output in the truth-table. Enter 0s elsewhere.
2. Encircle the octets, quads and pairs. Remember to roll and overlap to get the largest groups possible.
3. If any isolated 1s remain, encircle each.
4. Eliminate any redundant group.
5. Write the Boolean equation by ORing the products corresponding to the encircled groups.

2.3 Quine-McCluskey Method:

The K-map can be expanded to represent functions having more than five variables. The minimization process becomes increasingly difficult as the size of the map increases, however, since the various groupings becomes more and more disjointed. When working with larger functions, the tabular reduction method developed by Quine and modified by McCluskey is an alternative to the K-map method. The Quine-McCluskey minimization method involves simple, repetitive operations that compare each minterm that is present in a sum-of-minterms expression for a Boolean function to all other minterms with which it may form a combinable grouping. The reduced implicant that result from this process are compared among themselves to identify further reductions

until no further reductions are possible. This process identifies all the prime-implicants that may be used to represent the function in simplified SOP form. A selection process is then required to determine which set of prime-implicants does the best job of covering the function.

2.3.1 Cubical Representation Of Boolean Functions:

We are all familiar with the notion of geometric representation of a continuous variable as a distance along a straight line.

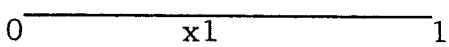


fig 2.3

In a similar fashion, a switching variable , which can take only two values, can be represented by two points at the ends of a single line(fig 2.3). We extend this to represent two variables by points in a plane. Similarly, the four possible values of two switching variables can be represented by the four vertices of a square(fig 2.4). The extension to three is shown in (fig 2.5).

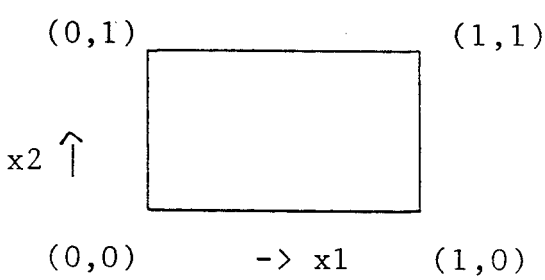


fig 2.4

The extension to more than three variables, requiring figures of more than three dimensions, is

geometrically difficult but conceptually simple enough.

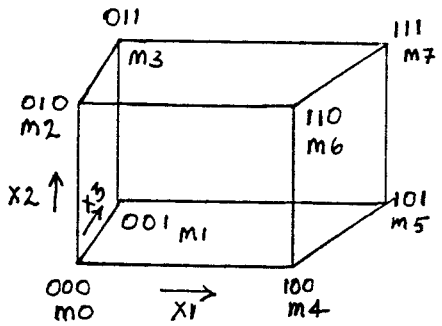


fig 2.5

In general, we say that we represent the various possible combinations of n-variables as points in n-space, and the collection of all 2^n possible points will be set to form the vertices of an n-cube, or a Boolean hypercube.

To represent funtions on the n-cube, we set up a one-to-one correspondence between the minterms of n-variables and the vertices of n-cube. These vertices, of a given function, may be referred to as the 0-cubes of the function. Two 0-cubes of a function are said to form a 1-cube, if they differ in only one co-ordinate. The 1-cubes may be denoted by placing an X in the coordinate having different values and darkening the line between the pair of 0-cubes(fig 2.6). In a similar fashion, a set of four 0-cubes, whose coordinate values are the same in all but two

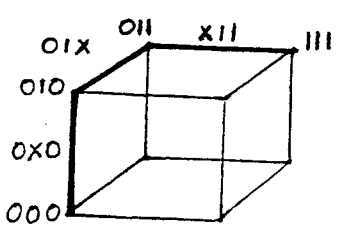


fig 2.6

variables, are said to form a 2-cube of the function. A 2-cube may be represented as a shaded plane(fig

2.7) shows the cubic representation of a function exhibiting five 0-cubes, five 1-cubes and one 2-cube.

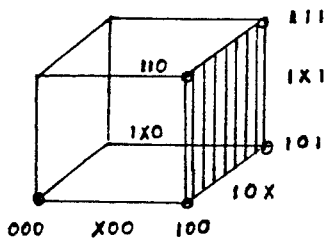


fig 2.7

2.3.2 Determination Of Prime-Implicants:

The goal of the Quine-McCluskey algorithm is a special case of the following.

OBJECT: Select the smallest possible subset, D, of aa set of objects, A, so that some criterion, f, is satisfied.

In the context of finding a minimal sum of products realization, set A is all possible Boolean cubes of any dimension that fall within a given Boolean function, f. The set D will consist of all products in the minimal sum of products realization. The algorithm for computing D is given in fig 2.8.

The first step, represented by blockI of fig 2.8 is the minterm list of the function. If the funtion is not of this form, it must be converted to this form by any of the methods. Assume the following function:

$f(A,B,C,D)= \Sigma\ m(0,2,3,6,7,8,9,10,13)$

Intially, find all possible 1-cubes of the function. A 1-cube is found by combining two 0-cubes, which are identical except in one variable.

* First, convert the minters to binary form and then count

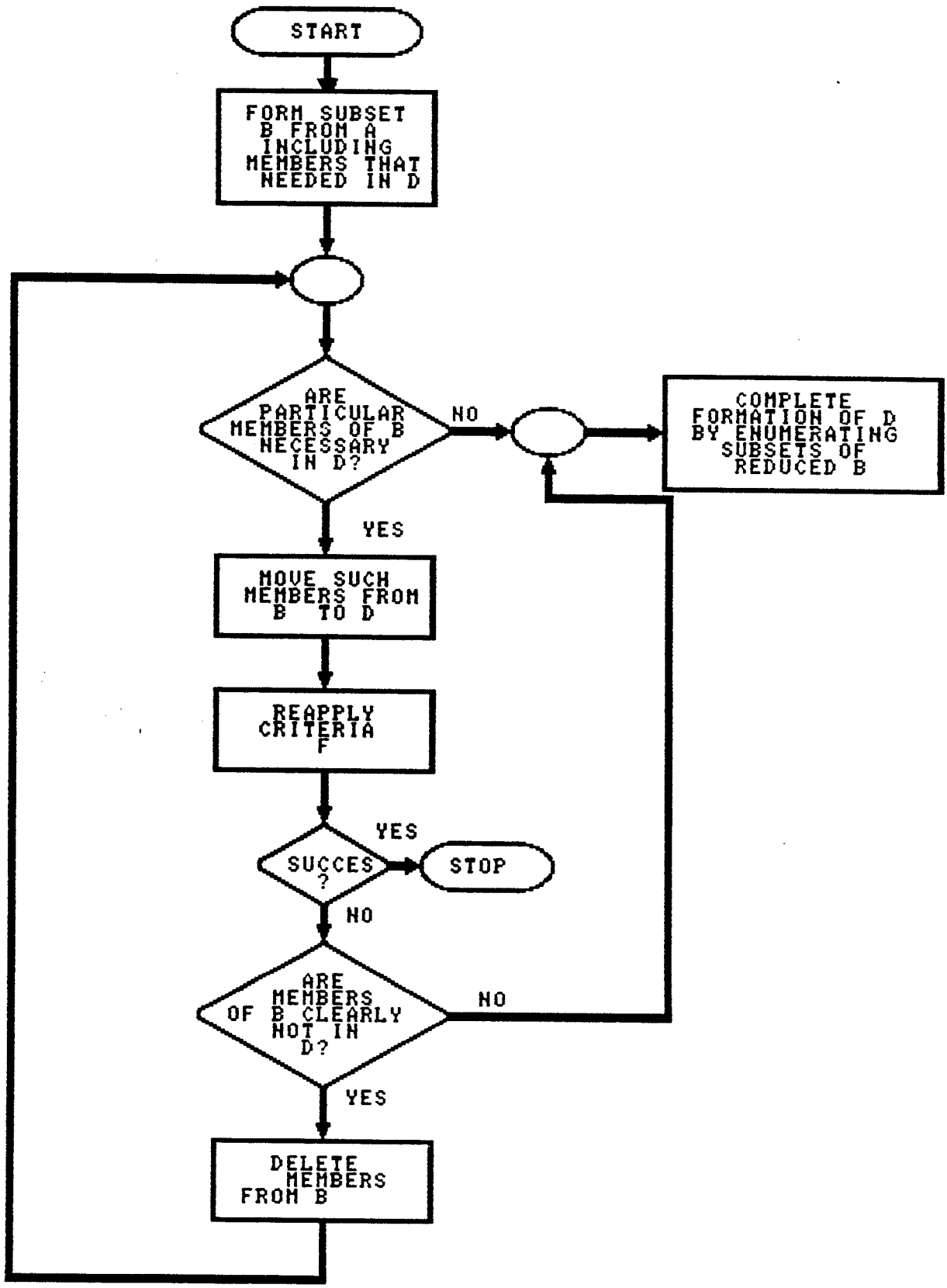

START
FORM SUBSET B FROM A INCLUDING MEMBERS THAT NEEDED IN D
ARE PARTICULAR MEMBERS OF B NECESSARY IN D?
NO
COMPLETE FORMATION OF D BY ENUMERATING SUBSETS OF REDUCED B
YES
MOVE SUCH MEMBERS FROM B TO D
REAPPLY CRITERIA F
YES
SUCCES ?
STOP
NO
ARE MEMBERS OF B CLEARLY NOT IN D?
NO
YES
DELETE MEMBERS FROM B

Fig 2.8

the number of 1s in the binary representation, as shown in fig 2.9

Minterm	Binary form	No: of 1s
m0	0000	0
m2	0010	1
m3	0011	2
m6	0110	2
m7	0111	3
m8	1000	1
m9	1001	2
m10	1010	2
m13	1101	3

fig 2.9

* Reorder the minterm list in accordance with the number of 1s in the binary representation(fig 2.10a). Separate the minterms into groups with the same number of 1s by horizontal lines.

1s	Minterms	
0	m0	0000
1	m2	0010
	m8	1000
2	m3	0011
	m6	0110
	m9	1001
	m10	1010
3	m7	0111
	m13	1101

(a)

1-cubes	
0,2	00x0
0,8	x000
2,3	001x
2,6	0x10
2,10	x010
*8,9	100x
8,10	10x0
3,7	0x11
6,7	011x
*9,13	1x01

(b)

2-cubes	
*0,2,8,10	x0x0
*2,3,6,7	0x1x

(c)

figure 2.10

* Determine the 1-cube, If two minterms are to combine, the binary representation must be identical except in one position, in which one minterm will have a 0, the

other a 1. The latter has one more 1 than the former, so that the two minterms must be in adjacent groups in the list.

* If two minterms are the same in every position but one, place a check to the right of both minterms to show that they have been covered and enter the 1-cube in the next column(fig 2.10b). The 1-cube and the decimal numbers of the combining minterms are listed.

* Every 1-cube must be found, although there is no-need to check off a minterm more than once.

* Next step is a search for possible combination of pairs of 1-cubes into 2-cubes. Therefore two 1-cubes are checked off. Unchecked entries -8,9 and 9,13 are marked with an asterisk(*) to indicate that they are prime-implicants.

* To find a minimal sum, we construct a prime-implicant table (fig 2.11). Each column corresponds to a minterm. At the left of each row are listed the prime implicant of the function arranged in groups (according to number of litera;s in the product). There is also an extra column on the left and an extra row at the bottom.

* For each row (except the bottom), checks are placed in the columns corresponding to the minterms contained in the prime implicant listed on that row.

* The completed prime implicant table is then inspected for columnns containing only a single check mark. This indicates that the corresponding prime implicant is the

Marking Min_Terms

		0	2	3	6	7	8	9	10	13
	0,2,8,10	✓	✓				✓		✓	
	2,3,6,7		✓	✓	✓	✓				
	8,9						✓	✓		
	9,13							✓		✓

fig 2.11

Checking For Essenial Pims...

		0	2	3	6	7	8	9	10	13
*	0,2,8,10	✓	✓				✓		✓	
*	2,3,6,7		✓	✓	✓	✓				
	8,9						✓	✓		
*	9,13							✓		✓
		✓	✓	✓	✓	✓	✓	✓	✓	✓

fig 2.12

only one containing the literal and must be included in the sum-of-product(SOP) realization. We therefore say that it is an essential prime implicant and mark it with an asterisk in the leftmost column. We also place checks in the bottom row in the columns to indicate that those literals are included in an already selected prime-implicants

* When the search for essential prime implicants is completed we inspect the bottom row to see if all columns have been checked(fig 2.12). If so, all the 0-cubes are contained in the essential prime implicants anfd the sum of essential prime implicants is the minimal SOP realization.

* The final step is to determine the actual products. The difference numbers for each prime-implicant indicates the variables that have been eliminated.

* Write out the binary equivalent of any one of the minterms in the prime implicant, cross out the positions having the binary weights in parenthesis and convert the remaining digits to the appropriate literals.

0,2,8,10(2,8) = 0000 -> bd

2,3,6,7 (1,4) = 0010 -> aC

9,13 (4) = 1001 -> AcD

Thus, the minimal sum of products is

$$f(A,B,C,D) = \Sigma\, m(0,2,3,6,7,8,9,10,13)$$

$$= Bd + aC + AcD$$

In the above example, the prime implicants covered all the minterms. This is not always the case. For the function given below, the essential prime implicants do not cover all minterms.

$f(A,B,C,D,E) = \sum m(1,2,3,5,9,10,11,18,19,20,21,23,25,26,27)$

when we select the essential prime implicants on the table, we find that they do not cover all the minterms of the function. we must apply some cost criteria in making a selection from the remaining non essential terms.

To make it easier to find the apppropriate terms a reduced prime implicant table is set up, listing only the 0-cubes not contained in the essential prime implicants as columns and those non essential prime implicants containing any of the uncovered 0-cubes as rows. In the reduced table, we note that ther could be no possible advantage to using e, since it covers only m5, while d which has the same cost, covers both m5 and m1. Thus removing e from the table cannot prevent us from finding a minimal sum.

We next note that g and h cover the same minterm at the same cost. i.e., g and h cover m23. Thus,removing either g or h from the table cannot prevent us from finding the minimal sum. In this problem we remove h. The rows e and h are removed as called by blockIV of the flowchat (fig 2.8) to form a still further reduced table, as shown in the 2.13

On this table, it can be seen that d is the only prime implicant left to cover the minterm m5 and g is another

Checking For Essenial Pins...:

		1	2	3	5	9	10	11	18	19	20	21	23	25	26	27
*	2,3,10,11,18,19,26,27		✓	✓			✓	✓	✓	✓					✓	✓
	1,3,9,11	✓		✓		✓		✓								
*	9,11,25,27					✓		✓						✓		✓
	1,5	✓			✓											
	5,21				✓							✓				
*	20,21										✓	✓				
	19,23									✓			✓			
	21,23											✓	✓			
			✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓	✓

fig 2.13 a

prime implicant, which is the only one left to cover the minterm m23. In the table, we therefore, mark them with double asterisk(**) to indicate that they are the secondary essential prime implicants. The selection of the secondary essential prime implicants is another application of block II of (fig 2.8),i.e., on the second pass through the flow chart. Thus, on each pass through the flow chart, we obtain the essential prime implicants,

which were not obtained through the previous pass. We also note that these essential prime implicants in the given problem, cover all of the remaining minterms, so the process terminates with success on this second pass.

In this problem, the minimal sum is given by

$$f(A,B,C,D,E) = \underbrace{a + b + f}_{\text{essential}} + \underbrace{d + g}_{\text{secondary essential}}$$

$$= (2,3,10,11,18,19,26,27) + (9,11,25,27) + (20,21) + (1,5) + (19,23)$$

$$= cD + BcE + AbCd + abdE + AbDE$$

to formalize the above procedure, we state the following rules.

RULE 1: Two rows, a and b, of a reduced prime implicant table that cover the same minterms, i.e., have checks in exactly the same columns, are said to be interchangeable.

RULE 2: Given two rows, a and b, in a reduced prime

Checking For Essenial Pims...

		1	5	23
	1,3,9,11	✓		
*	1,5	✓	✓	
*	19,23			✓
		✓	✓	✓

fig 2.13b

implicant table, row a is said to dominate row b if row a has checks in all columns in which row b has checks and also has a check in atleast one column in which row b does not have a check.

One implementation of block V of (fig 2.8) will be used when no column has a single check and no row dominates any other row as shown in fig 2.14. In fig 2.14,from the first column, either a or f must be selected to cover m6, so we write

$$(a + f) = 1$$

similarly, either a or c must be selected to cover m7; so

$$(a + f)\ (a + c) = 1$$

Hence, we build a product of sums, each sum listing the prime implicants that contain a particular minterm. The complete expression is

$$(a+f)\ (a+c)\ (b+c)\ (e+f)\ (d+e)\ (b+d) = 1$$

	6	7	15	38	46	47
a	✓	✓				
b			✓			✓
c		✓	✓			
d					✓	✓
e				✓	✓	
f	✓			✓		

figure 2.14

Applying the distributive law, we obtain,

$$abe + abdf + acde + bcef + cdf = 1$$

To satisfy the above equation, at least one of the five products must be 1. Of these abe and cdf represent a covering of the table by three prime implicants. The total cost is least for a,b and e, so these three are included with any essential prime implicants in a minimal SOP.

The Quine-McCluskey method can be modified to handle don't care terms also.

CHAPTER III

SOFTWARE ROUTINES TO QUINE-McCLUSKEY METHOD

This chapter explains the flowchart and the algorithm of the various modules used in the minimzation procedure of the Quine-McCluskey method. The software which is written supports the don't care conditions.

3.1 Selection Of Prime-Implicants:

Initially, in this module to find the prime implicants, the number of variables are read. Then, the number of minterms which can be received are 2•number of variables. Among the number of minterms, read the number of 1s and the number of don't cares.

Convert the received minterms into its corresponding binary form, and count the number of 1s in their binary equivalent. The next step, is to sort the minterms depending on the number of 1s in the binary equivalent. Group the minterms which have equal number of 1s and separate the groups by having some table information.

Find the combination of the minterms which form a 1-cube. A 1-cube is a combination of two minterms which are identical except in one position. This indicates that the two minterms must be in adjacent groups in the list. If two minterms are the same in every position but one, it indicates that they have been covered and enter into the table of 1-

cubes. List the cubical form of the 1-cube with an'x' in the position that does not agree, and also list the decimal numbers of the combining minterms.

From the list of 1-cubes, find the combination of minterms which form the 2-cube. A 2-cube is a combination of two 1-cubes which are identical except in one position. Here again, it indicates that the two minterms have been covered and enter into the next table. While searching for 1-cubes and 2-cubes, if there are any minterms which are not covered, then they are the prime-implicants. Continue the above procedure until the final table is formed in which the two minterm combination cannot be combined to form another cube. Then, all the entries of the last table are prime-implicants which can be entered into the prime-implicant list.

3.2 Selection Of Essential Prime-Implicants:

After obtaining the prime-implicants, set the prime-implicant table for all the prime-implicant in the list. The prime-implicants having the same cost are put in same groups. For each prime-implicant, checks are placed in the columns corresponding to the minterms contained in the prime-implicant.

A procedure, essential prime-implicant, is written which will search for the essential prime-implicants in the prime-implicant table. This procedure inspects the column containing onluy a single check. Then, the minterm

combination corresponding to this column is an essential prime-implicant. Store this minterm combination in another separate table.

Continuing the search, we obtain all the essential prime-implicants. When the search is complete, we inspect the bottom row to see if all the columns have been checked. If so, then all the 0-cubes are contained in the essential prime-implicants and the sum of the essential prime-implicants in the minimal sum-of-product realization.

3.3 Selection Of Secondary Essential Prime-Implicants:

If the essential prime-implicants do not cover all the minterms, then separate procedure is written to reduce the prime-implicant table. In this procedure some cost criteria is applied. The reduced prime-implicant table contain the essential prime-implicants as the columns and those non-essential prime-implicants containing any of the uncovered 0-cubes as rows. In the reduced table the following definitions are applied.

1. Two rows of a reduced prime-implicant table, that cover the same minterms are said to be interchangeable.
2. Given two rows in a reduced prime-implicant table, one row is said to dominate another row, if the first row covers all the minterms the second row covers, and an extra minterm which the second row does not cover.

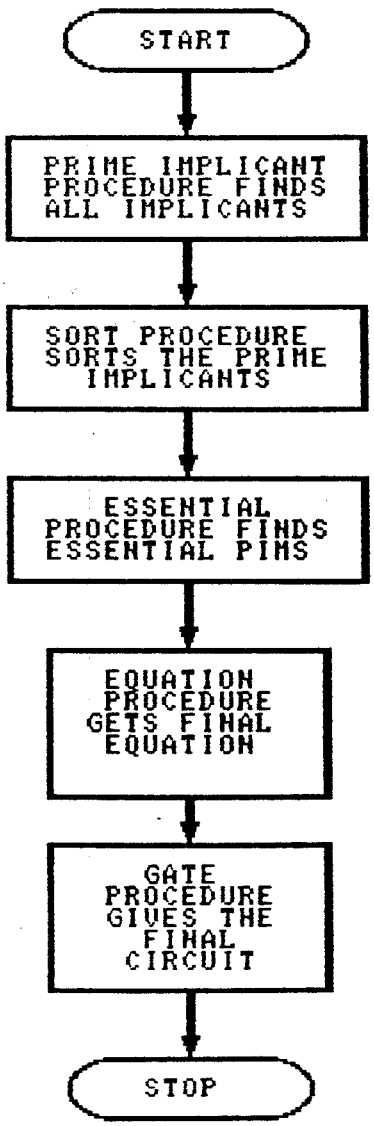
Using the above definitions, the reduced prime-

implicant table is further reduced and using the same procedure which was used to find the essential prime-implicants, the secondary essential prime-implicants are obtained from the reduced table, and the final equation is obtained.

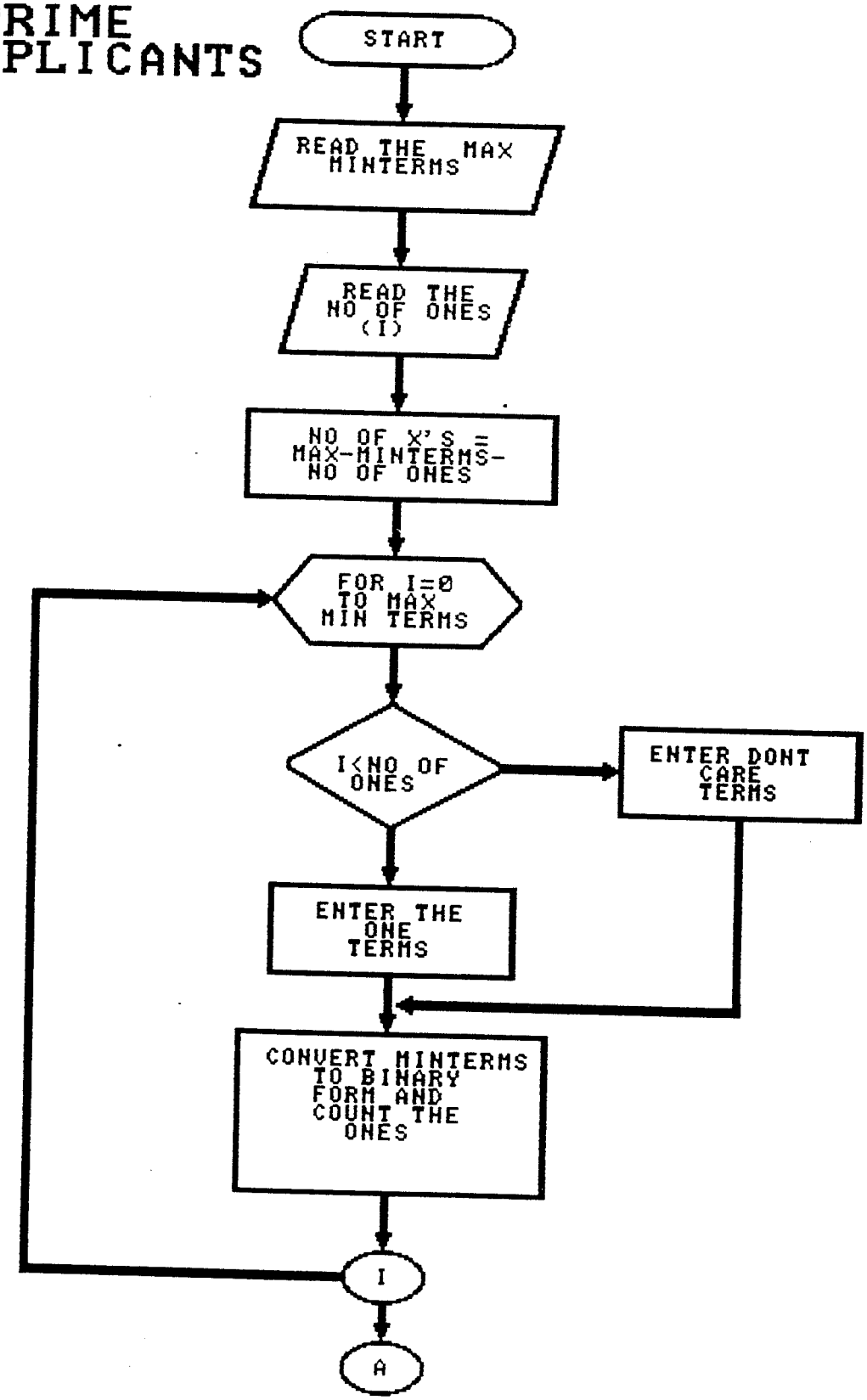
3.4 Circuit Drawing:

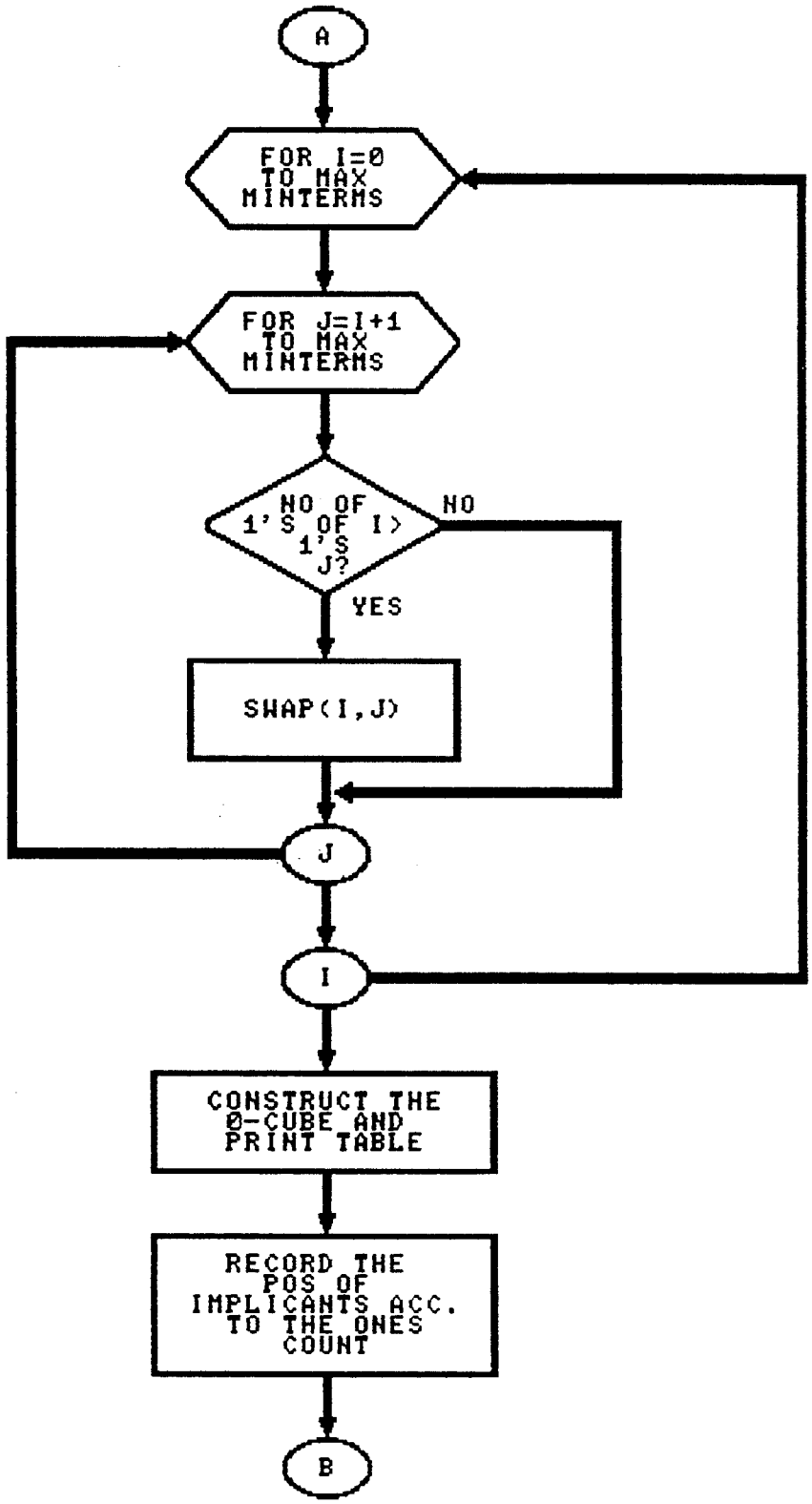
A separate procedure is written to draw the logical circuit for the reduced equation. The available graphic utilities in "C" are used to draw the logical circuit. The various gates are drawn using the various functions.

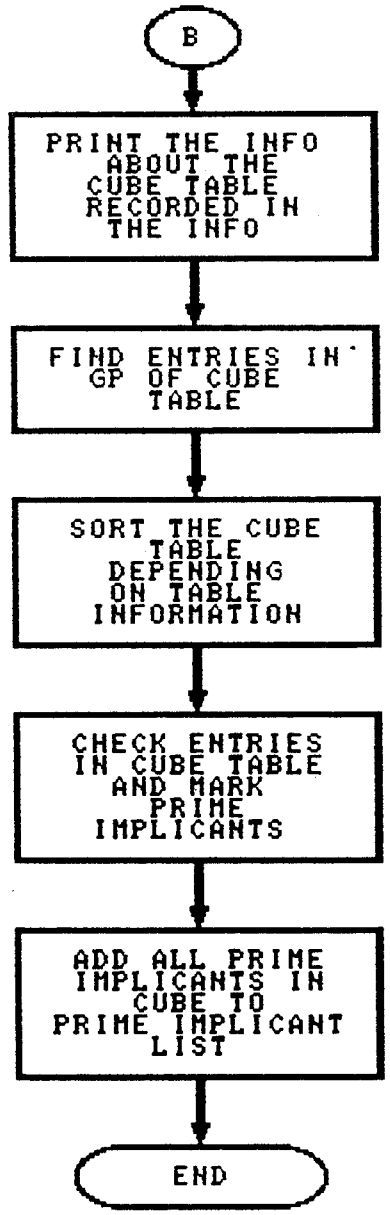
SYSTEM FLOW


START
PRIME IMPLICANT PROCEDURE FINDS ALL IMPLICANTS
SORT PROCEDURE SORTS THE PRIME IMPLICANTS
ESSENTIAL PROCEDURE FINDS ESSENTIAL PIMS
EQUATION PROCEDURE GETS FINAL EQUATION
GATE PROCEDURE GIVES THE FINAL CIRCUIT
STOP

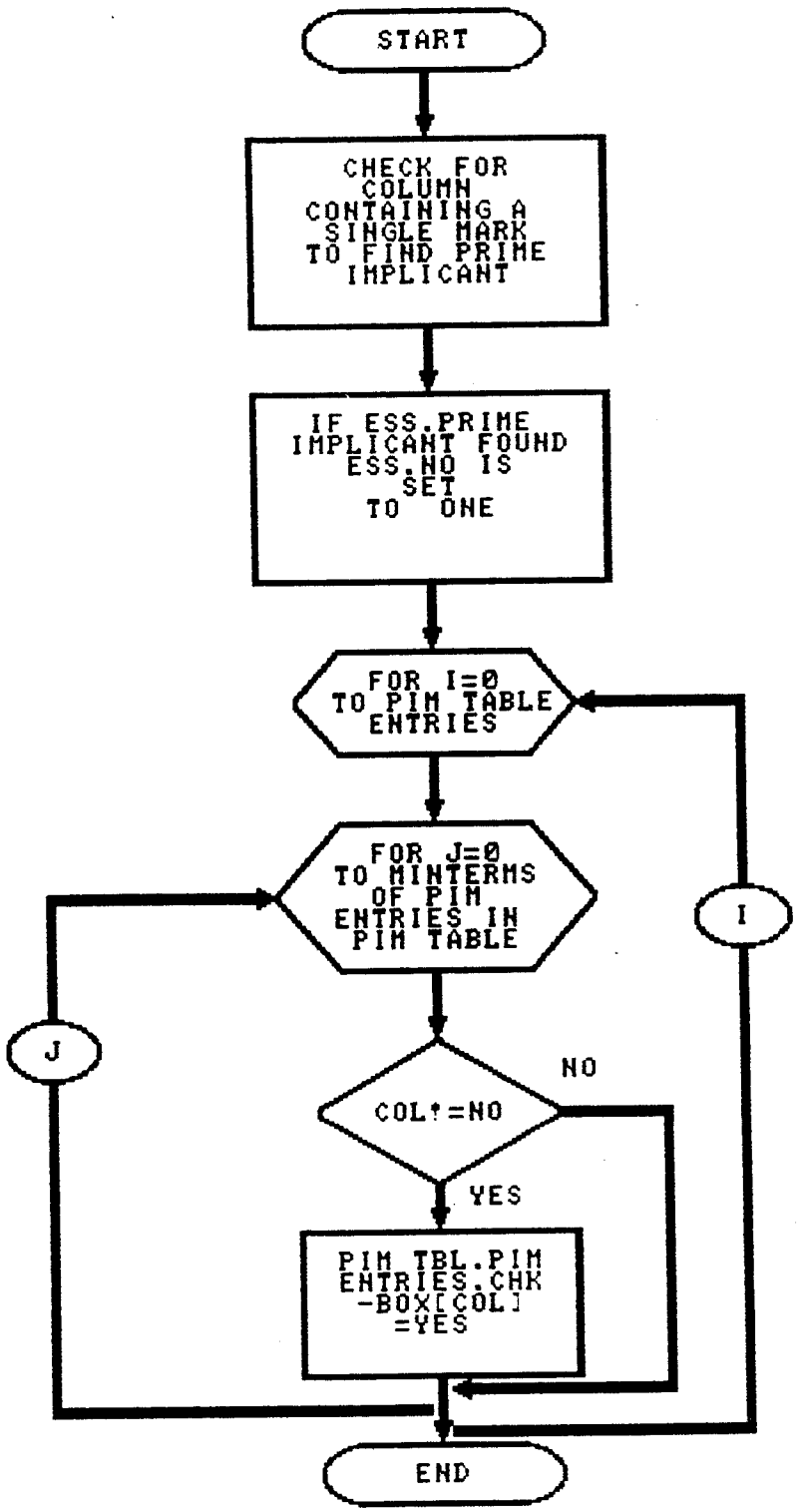
PRIME IMPLICANTS



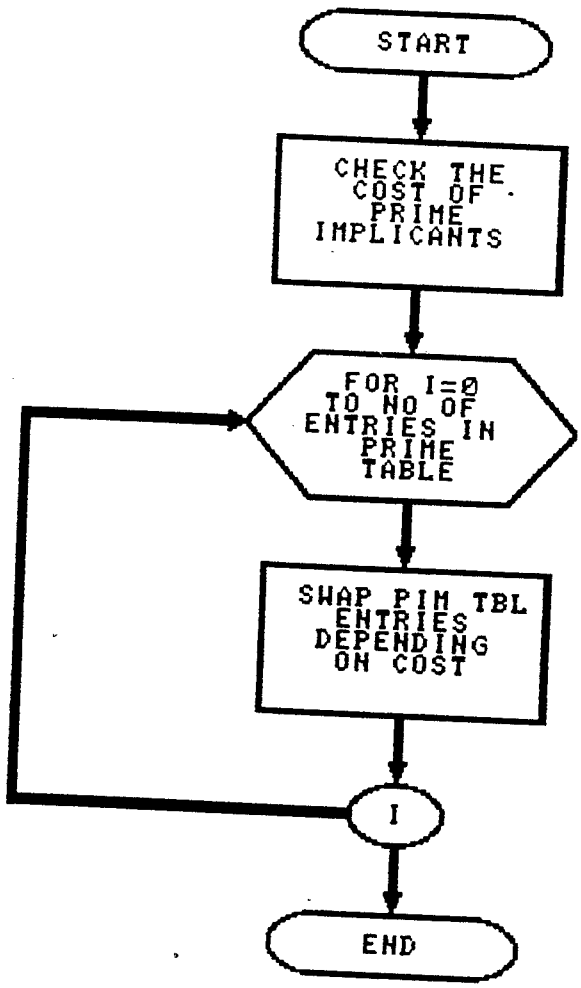

A
FOR I=0 TO MAX MINTERMS
FOR J=I+1 TO MAX MINTERMS
NO OF 1'S OF I > 1'S J?
NO
YES
SWAP(I,J)
J
I
CONSTRUCT THE 0-CUBE AND PRINT TABLE
RECORD THE POS OF IMPLICANTS ACC. TO THE ONES COUNT
B


B
PRINT THE INFO ABOUT THE CUBE TABLE RECORDED IN THE INFO
FIND ENTRIES IN GP OF CUBE TABLE
SORT THE CUBE TABLE DEPENDING ON TABLE INFORMATION
CHECK ENTRIES IN CUBE TABLE AND MARK PRIME IMPLICANTS
ADD ALL PRIME IMPLICANTS IN CUBE TO PRIME IMPLICANT LIST
END

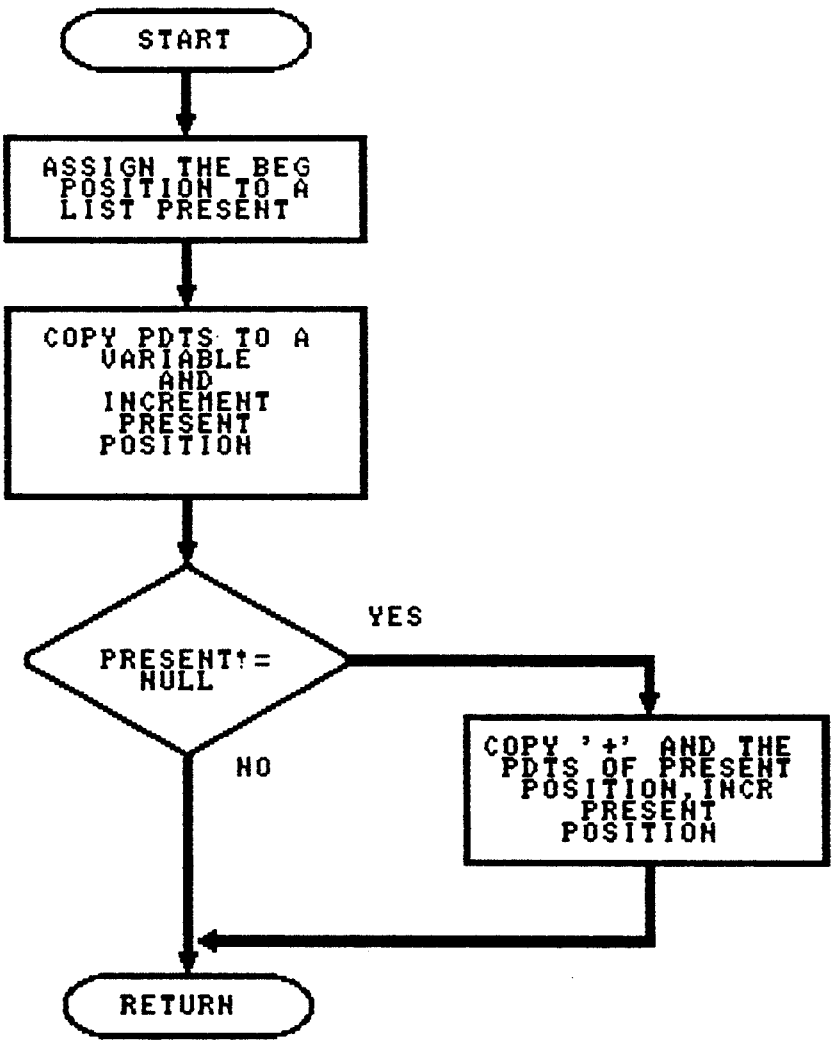
ESSENTIAL PRIME IMPLICANTS



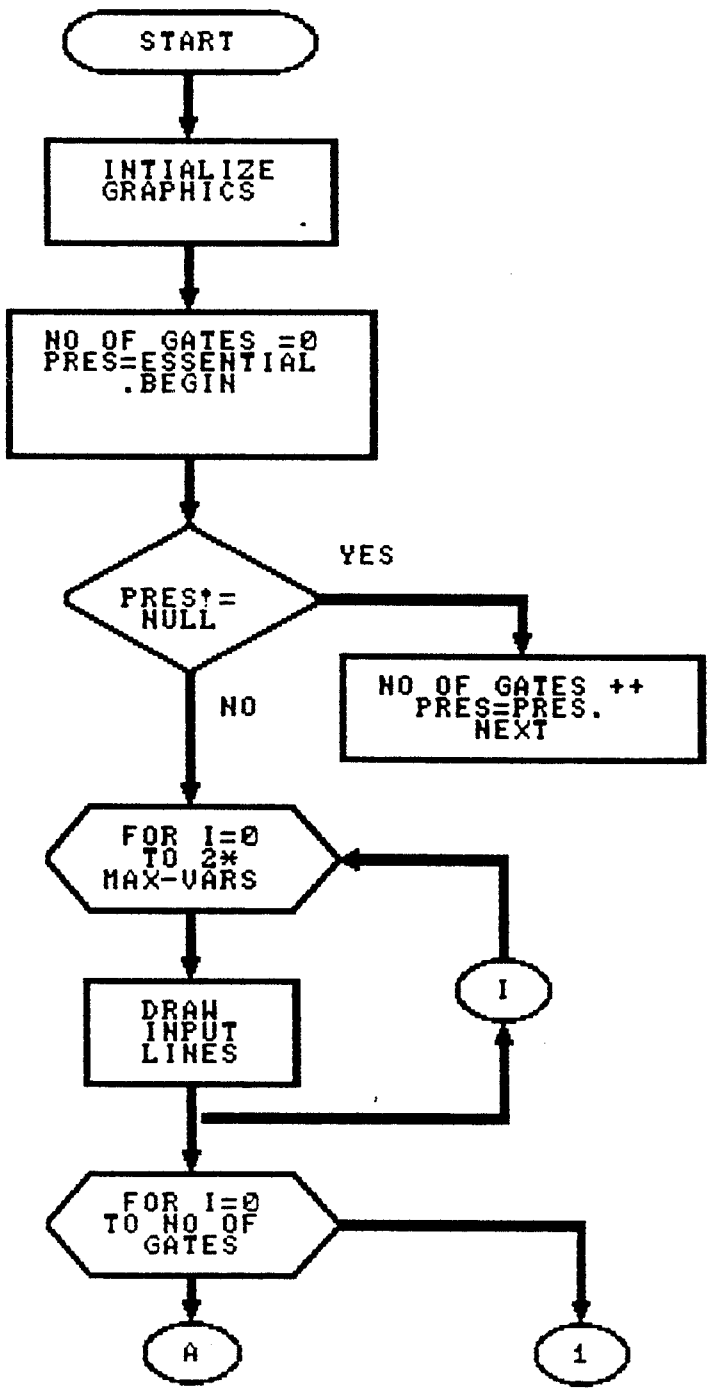
SORT

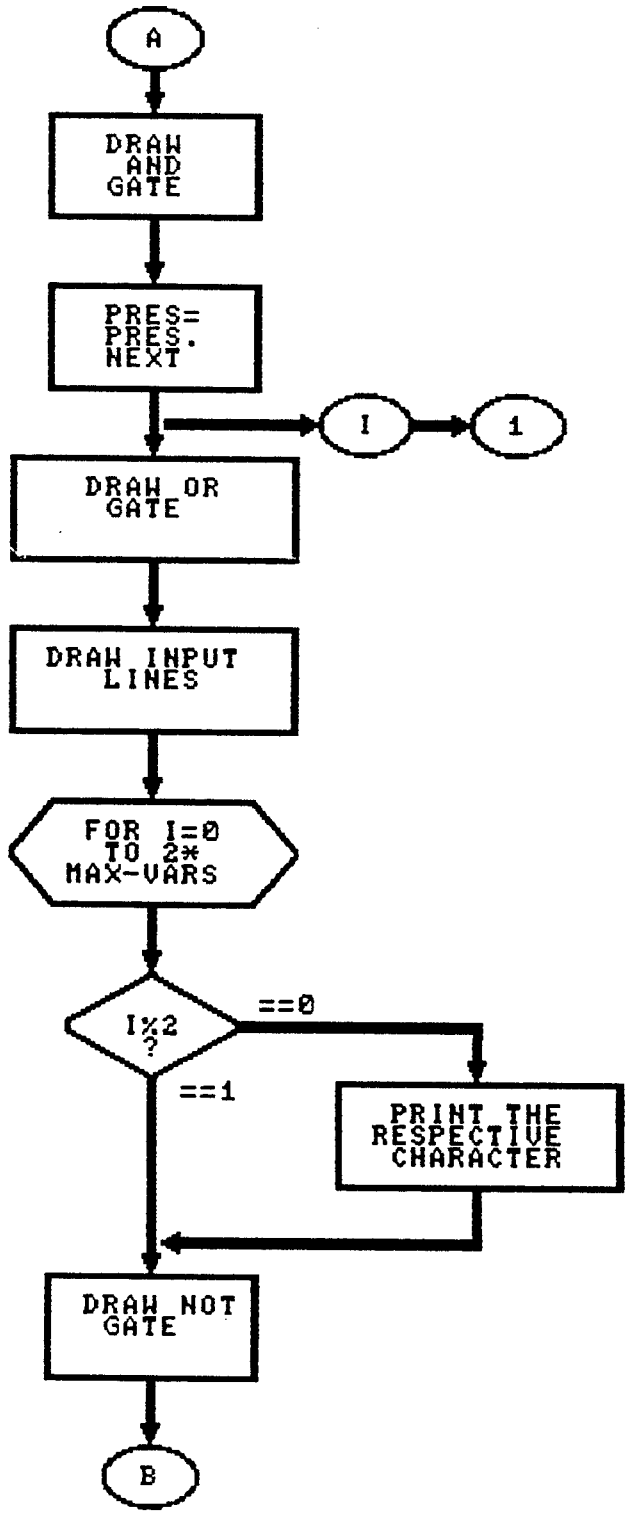

START
CHECK THE COST OF PRIME IMPLICANTS
FOR I=0 TO NO OF ENTRIES IN PRIME TABLE
SWAP PIM TBL ENTRIES DEPENDING ON COST
I
END

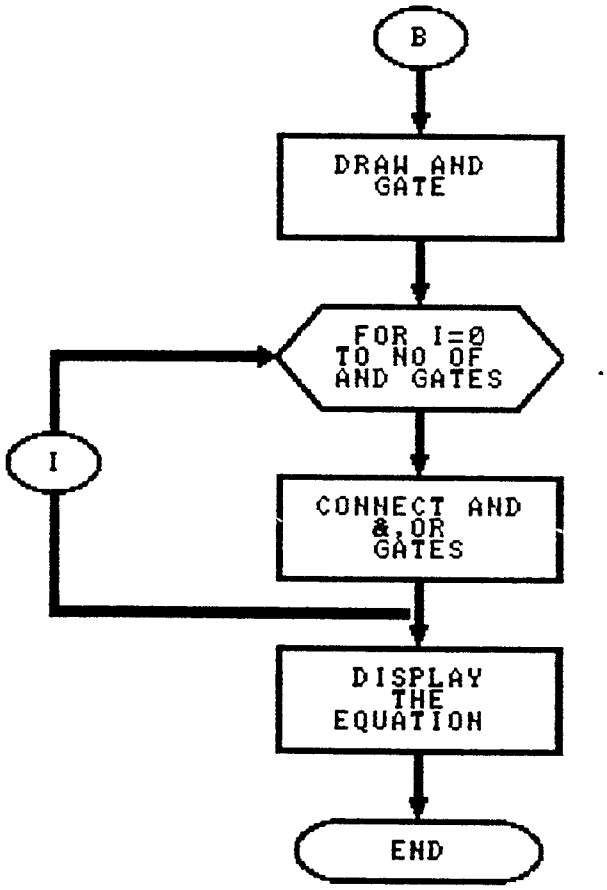
EQUATION


START
ASSIGN THE BEG POSITION TO A LIST PRESENT
COPY PDTS TO A VARIABLE AND INCREMENT PRESENT POSITION
PRESENT != NULL
YES
NO
COPY '+' AND THE PDTS OF PRESENT POSITION, INCR PRESENT POSITION
RETURN

GATE


START
INTIALIZE GRAPHICS
NO_OF_GATES =0
PRES=ESSENTIAL
.BEGIN
PRES!= NULL
YES
NO
NO_OF_GATES ++
PRES=PRES.
NEXT
FOR I=0
TO 2*
MAX-VARS
DRAW
INPUT
LINES
I
FOR I=0
TO NO OF
GATES
A
1


A
DRAW AND GATE
PRES= PRES. NEXT
I
1
DRAW OR GATE
DRAW INPUT LINES
FOR I=0 TO 2* MAX-VARS
I%2 ?
==0
==1
PRINT THE RESPECTIVE CHARACTER
DRAW NOT GATE
B


B
DRAW AND GATE
FOR I=0 TO NO OF AND GATES
I
CONNECT AND &,OR GATES
DISPLAY THE EQUATION
END

```
Enter the number of variables:   5
Enter the maximum minterms    :  14
Enter the number of 1 terms   :  14
  0..1 term # 0   :  0
  1..1 term # 1   :  6
  2..1 term # 2   :  8
  3..1 term # 3   :  10
  4..1 term # 4   :  12
  5..1 term # 5   :  14
  6..1 term # 6   :  17
  7..1 term # 7   :  19
  8..1 term # 8   :  20
  9..1 term # 9   :  22
 10..1 term # 10  :  25
 11..1 term # 11  :  27
 12..1 term # 12  :  28
 13..1 term # 13  :  30
```

Table just Constructed from minterms

0-Cubes		
*	0	00000
*	8	01000
*	6	00110
*	10	01010
*	12	01100
*	17	10001
*	20	10100
*	19	10011
*	14	01110
*	22	10110
*	25	11001
*	28	11100
*	27	11011
*	30	11110

After Sorting according to cost

0-Cubes		
*	0	00000
*	8	01000
*	6	00110
*	10	01010
*	12	01100
*	17	10001
*	20	10100
*	14	01110
*	19	10011
*	22	10110
*	25	11001
*	28	11100
*	27	11011
*	30	11110

After Checking For Prime Implicants

0-Cubes		
√	0	00000
√	8	01000
√	6	00110
√	10	01010
√	12	01100
√	17	10001
√	20	10100
√	14	01110
√	19	10011
√	22	10110
√	25	11001
√	28	11100
√	27	11011
√	30	11110

The Next Table

1-Cubes		
*	0,8	0x000
*	8,10	010x0
*	8,12	01x00
*	6,14	0x110
*	6,22	x0110
*	10,14	01x10
*	12,14	011x0
*	12,28	x1100
*	17,19	100x1
*	17,25	1x001
*	20,22	101x0
*	20,28	1x100
*	14,30	x1110
*	19,27	1x011
*	22,30	1x110
*	25,27	110x1
*	28,30	111x0

After Checking For Prime Implicants

1-Cubes		
*	0,8	0x000
√	8,10	010x0
√	8,12	01x00
√	6,14	0x110
√	6,22	x0110
√	10,14	01x10
√	12,14	011x0
√	12,28	x1100
√	17,19	100x1
√	17,25	1x001
√	20,22	101x0
√	20,28	1x100
√	14,30	x1110
√	19,27	1x011
√	22,30	1x110
√	25,27	110x1
√	28,30	111x0

The Next Table

	2-Cubes	
*	8,10,12,14	01xx0
*	6,14,22,30	xx110
*	12,14,28,30	x11x0
*	17,19,25,27	1x0x1
*	20,22,28,30	1x1x0

```
No Of Prime Implicants := 6
 0,8                                   0x000
 8,10,12,14                            01xx0
 6,14,22,30                            xx110
 12,14,28,30                           x11x0
 17,19,25,27                           1x0x1
 20,22,28,30                           1x1x0
```

Fresh Table Constructed With PIMs

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	0,8														
	8,10,12,14														
	6,14,22,30														
	12,14,28,30														
	17,19,25,27														
	20,22,28,30														

After Reversing The Table

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	20,22,28,30														
	17,19,25,27														
	12,14,28,30														
	6,14,22,30														
	8,10,12,14														
	0,8														

After Sorting Table

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	8,10,12,14														
	17,19,25,27														
	12,14,28,30														
	6,14,22,30														
	20,22,28,30														
	0,8														

Marking Min_Terms

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	8,10,12,14			✓	✓	✓	✓								
	17,19,25,27							✓	✓			✓	✓		
	12,14,28,30					✓	✓							✓	✓
	6,14,22,30		✓				✓				✓				✓
	20,22,28,30									✓	✓			✓	✓
	0,8	✓		✓											

After Rule#1

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	8,10,12,14			✓	✓	✓	✓								
	17,19,25,27							✓	✓			✓	✓		
	12,14,28,30					✓	✓							✓	✓
	6,14,22,30		✓				✓				✓				✓
	20,22,28,30									✓	✓			✓	✓
	0,8	✓		✓											

After Rule#2

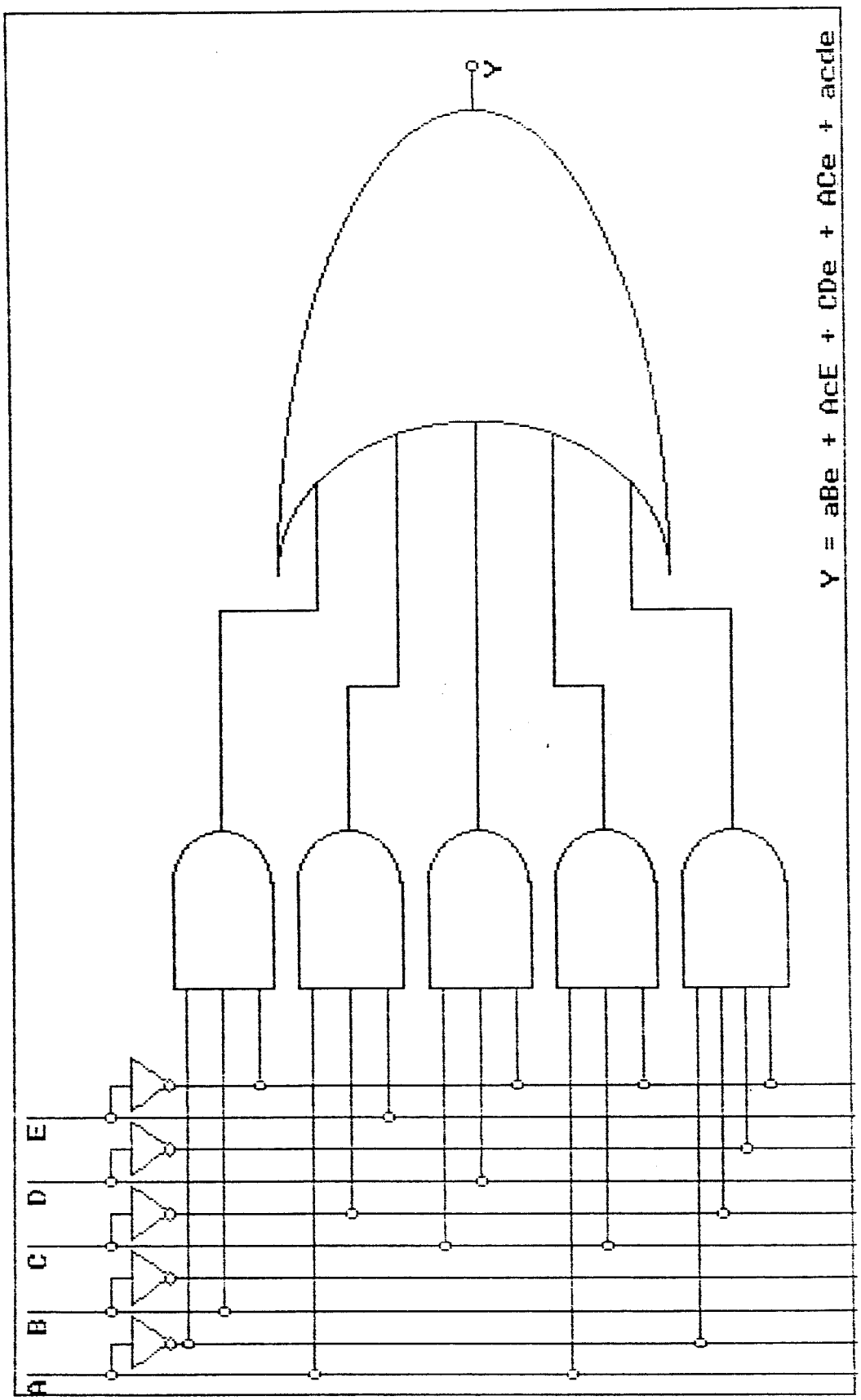
		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	8,10,12,14			✓	✓	✓	✓								
	17,19,25,27							✓	✓			✓	✓		
	12,14,28,30					✓	✓							✓	✓
	6,14,22,30		✓				✓				✓				✓
	20,22,28,30									✓	✓			✓	✓
	0,8	✓		✓											

Reduced Pin_Table

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
	8,10,12,14			✓	✓	✓	✓								
	17,19,25,27							✓	✓			✓	✓		
	12,14,28,30					✓	✓							✓	✓
	6,14,22,30		✓				✓				✓				✓
	20,22,28,30									✓	✓			✓	✓
	0,8	✓		✓											

Checking For Essenial Pins...

		0	6	8	10	12	14	17	19	20	22	25	27	28	30
*	8,10,12,14			✓	✓	✓	✓								
*	17,19,25,27							✓	✓			✓	✓		
	12,14,28,30					✓	✓							✓	✓
*	6,14,22,30		✓				✓				✓				✓
*	20,22,28,30									✓	✓			✓	✓
*	0,8	✓		✓											
		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓


Y = aBe + AcE + CDe + ACe + acde
Y
A
B
C
D
E

```
Enter the number of variables :  5
Enter the maximum minterms    : 14
Enter the number of 1 terms   :  9
Enter the number of X terms   :  5
  0..1 term # 0  :  1
  1..1 term # 1  :  4
  2..1 term # 2  :  7
  3..1 term # 3  : 14
  4..1 term # 4  : 17
  5..1 term # 5  : 20
  6..1 term # 6  : 21
  7..1 term # 7  : 22
  8..1 term # 8  : 23
  9..X term # 0  :  0
 10..X term # 1  :  3
 11..X term # 2  :  6
 12..X term # 3  : 19
 13..X term # 4  : 30
```

Table just Constructed from minterms

0-Cubes		
*	0	00000
*	4	00100
*	1	00001
*	20	10100
*	17	10001
*	3	00011
*	6	00110
*	22	10110
*	7	00111
*	14	01110
*	21	10101
*	19	10011
*	23	10111
*	30	11110

After Sorting according to cost

0-Cubes		
*	0	00000
*	1	00001
*	4	00100
*	3	00011
*	6	00110
*	17	10001
*	20	10100
*	7	00111
*	14	01110
*	19	10011
*	21	10101
*	22	10110
*	23	10111
*	30	11110

After Checking For Prime Implicants

0-Cubes		
√	0	00000
√	1	00001
√	4	00100
√	3	00011
√	6	00110
√	17	10001
√	20	10100
√	7	00111
√	14	01110
√	19	10011
√	21	10101
√	22	10110
√	23	10111
√	30	11110

The Next Table

1-Cubes		
*	0,1	0000x
*	0,4	00x00
*	1,3	000x1
*	1,17	x0001
*	4,6	001x0
*	4,20	x0100
*	3,7	00x11
*	3,19	x0011
*	6,7	0011x
*	6,14	0x110
*	6,22	x0110
*	17,19	100x1
*	17,21	10x01
*	20,21	1010x
*	20,22	101x0
*	7,23	x0111
*	14,30	x1110
*	19,23	10x11
*	21,23	101x1
*	22,23	1011x
*	22,30	1x110

After Checking For Prime Implicants

1-Cubes		
*	0,1	0000x
*	0,4	00x00
√	1,3	000x1
√	1,17	x0001
√	4,6	001x0
√	4,20	x0100
√	3,7	00x11
√	3,19	x0011
√	6,7	0011x
√	6,14	0x110
√	6,22	x0110
√	17,19	100x1
√	17,21	10x01
√	20,21	1010x
√	20,22	101x0
√	7,23	x0111
√	14,30	x1110
√	19,23	10x11
√	21,23	101x1
√	22,23	1011x
√	22,30	1x110

The Next Table

2-Cubes		
*	1,3,17,19	x00x1
*	4,6,20,22	x01x0
*	3,7,19,23	x0x11
*	6,7,22,23	x011x
*	6,14,22,30	xx110
*	17,19,21,23	10xx1
*	20,21,22,23	101xx

```
No Of Prime Implicants := 9
 0,1                                0000x
 0,4                                00x00
 1,3,17,19                          x00x1
 4,6,20,22                          x01x0
 3,7,19,23                          x0x11
 6,7,22,23                          x011x
 6,14,22,30                         xx110
 17,19,21,23                        10xx1
 20,21,22,23                        101xx
```

Fresh Table Constructed With Prime implicants

		1	4	7	14	17	20	21	22	23
	0,1									
	0,4									
	1,3,17,19									
	4,6,20,22									
	3,7,19,23									
	6,7,22,23									
	6,14,22,30									
	17,19,21,23									
	20,21,22,23									

After Reversing The Table

		1	4	7	14	17	20	21	22	23
	20,21,22,23									
	17,19,21,23									
	6,14,22,30									
	6,7,22,23									
	3,7,19,23									
	4,6,20,22									
	1,3,17,19									
	0,4									
	0,1									

Marking Min_Terms

		1	4	7	14	17	20	21	22	23
	4,6,20,22		✓				✓		✓	
	1,3,17,19	✓				✓				
	6,14,22,30				✓				✓	
	6,7,22,23			✓					✓	✓
	3,7,19,23			✓						✓
	20,21,22,23						✓	✓	✓	✓
	17,19,21,23					✓		✓		✓
	0,4		✓							
	0,1	✓								

After Removing Allx prime implicants

		1	4	7	14	17	20	21	22	23
	4,6,20,22		✓				✓		✓	
	1,3,17,19	✓				✓				
	6,14,22,30				✓				✓	
	6,7,22,23			✓					✓	✓
	3,7,19,23			✓						✓
	20,21,22,23						✓	✓	✓	✓
	17,19,21,23					✓		✓		✓
	0,4		✓							
	0,1	✓								

After Rule#1

		1	4	7	14	17	20	21	22	23
	4,6,20,22		✓				✓		✓	
	1,3,17,19	✓				✓				
	6,14,22,30				✓				✓	
	6,7,22,23			✓					✓	✓
	3,7,19,23			✓						✓
	20,21,22,23						✓	✓	✓	✓
	17,19,21,23					✓		✓		✓
	0,4		✓							
	0,1	✓								

After Rule#2

		1	4	7	14	17	20	21	22	23
	4,6,20,22		✓				✓		✓	
	1,3,17,19	✓				✓				
	6,14,22,30				✓				✓	
	6,7,22,23			✓					✓	✓
	20,21,22,23						✓	✓	✓	✓
	17,19,21,23					✓		✓		✓

Reduced Pin_Table

		1	4	7	14	17	20	21	22	23
	4,6,20,22		✓				✓		✓	
	1,3,17,19	✓				✓				
	6,14,22,30				✓				✓	
	6,7,22,23			✓					✓	✓
	20,21,22,23						✓	✓	✓	✓
	17,19,21,23					✓		✓		✓

Checking For Essenial Pins...

		1	4	7	14	17	20	21	22	23
*	4,6,20,22		✓				✓		✓	
*	1,3,17,19	✓				✓				
*	6,14,22,30				✓				✓	
*	6,7,22,23			✓					✓	✓
	20,21,22,23						✓	✓	✓	✓
	17,19,21,23					✓		✓		✓
		✓	✓	✓	✓	✓	✓		✓	✓

Next Table

		21
	20,21,22,23	
	17,19,21,23	

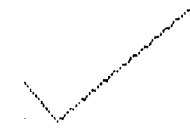
Marking Essential Prime Implicants..

		21
	20,21,22,23	✓
	17,19,21,23	✓

After Removing All× prime implicants

		21
	20,21,22,23	✓
	17,19,21,23	✓

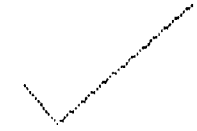
After Rule41

		21
	20,21,22,23	

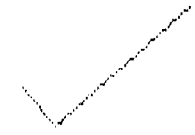
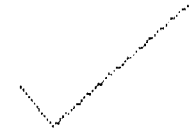
After Rule#2

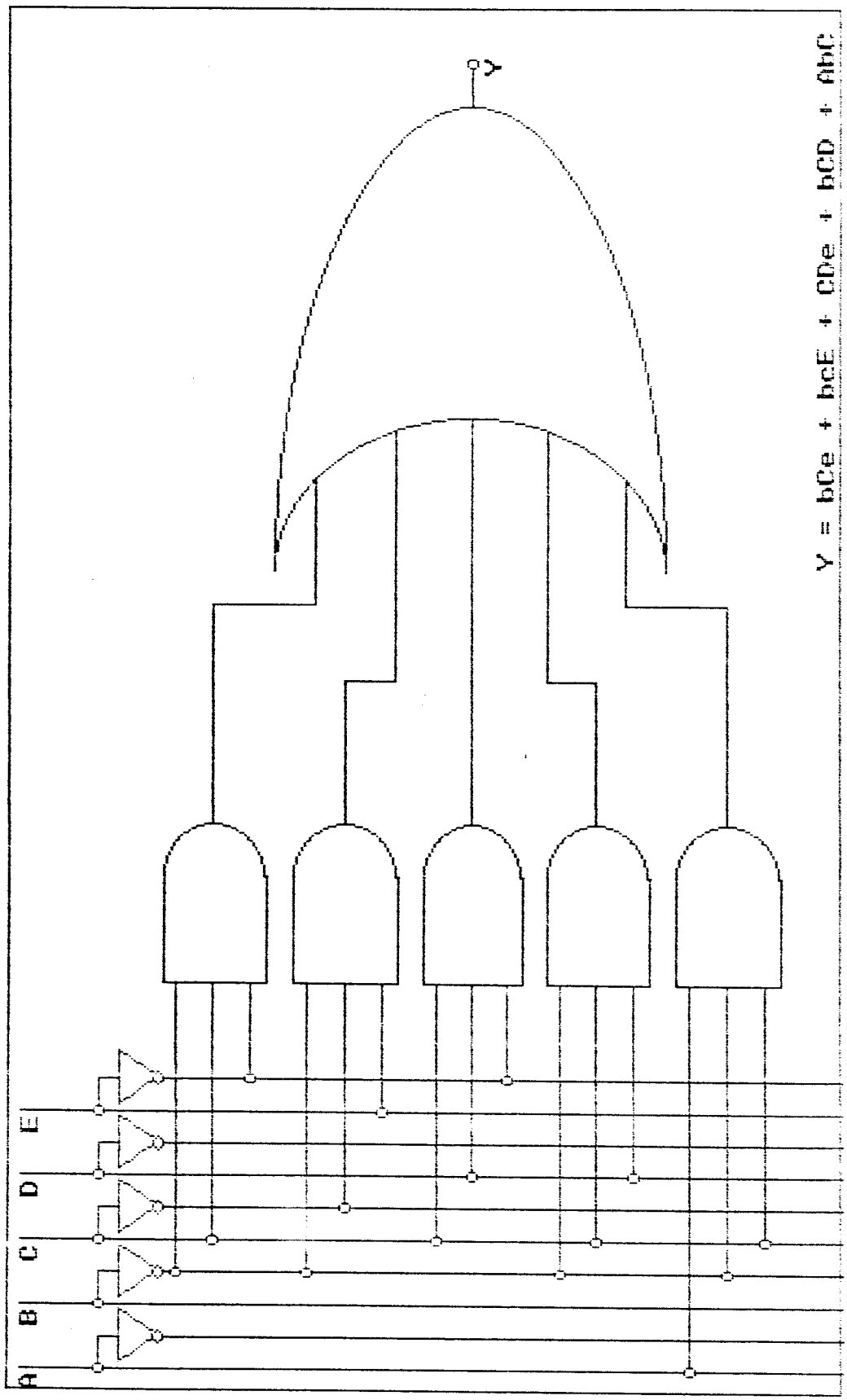
		21
	20,21,22,23	✓

Reduced Prime Implicant Table

		21
	20,21,22,23	

Checking For Essenial Pins...

		21
*	20,21,22,23	
		


Y = bCe + bcE + CDe + bCD + AbC
Y
A
B
C
D
E

CHAPTER V

CONCLUSION

An earnest attempt has been made by us, in minimising a given set of minterms to a reduced equation by switching-function-minimization method, i.e., Quine-McCluskey method.

By implementing the various graphic utilities present in "C", the logical circuit is drawn for the reduced equation using various gates.

Our project is generalised for a maximum of six variables. This project can also be extended for any number of variables, as the number of variables increases,the software must carefully calculate the size of various gates involved in the circuit drawing such that the gates in the logical circuit does not overlap.

The software that has been written by us for the minimization method can be run on IBM compatible PCs.

CHAPTER VI

FUTURE ENHANCEMENTS

The **Quine-McCluskey** method can be further extended to the minimization of any number of variables. This method can also be adapted to product-of-sum design, by designing of the zeros instead of the ones in the function. The function minimization of multiple output circuits can also be done using this method based on certain cost criteria.

This project can also be extended to circuits that contain memory elements, i.e., sequential circuits. The Quine-McCluskey method can be further applied to various problems in digital system design even in an integrated circuit environment.

CHAPTER VII

REFRENCES

1. FREDERICK J. HILL and GERALD R. PETERSON, " **Introduction To Switching Theory And Logical Design**",(Third Edition), Wiley, New york.

2. V. THOMAS RHYNE, "**Fundamentals Of Digital Systems Design**", Prentice-Hall, New Jersey.

3. SAMUEL C. LEE , "**Digital Circuits And Logic Design**", Prentice-Hall, India.

4. MITCHELL P. MARCUS, "**Switching Circuits For Engineers**", (Third Edition), Prentice-Hall, New Jersey.

5. BARTEE, T. C. " **Automatic Design Of Logical Networks**".

6. McCLUSKEY, E.J. "**Minimization Of Boolean Funtions**".

7. PETRICK, S.R. "**On The Minimization Of Boolean Functions**".

```
/*
maximum number of variables upto which this program can find
minimal SOP realization.
*/
#define MAX_VARS        6
#define MAX_POW2        64

/*
        Constants used in the program to improve readability.
*/
#define NO              (-111)
#define YES             100
#define NEXT            (-1000)
#define ALL             (-1111)

/*
        Standard Include Files.
*/
#include <math.h>
#include <stdio.h>
#include <alloc.h>
#include <graphics.h>

/*
        Structure definitions.
*/
struct IMPLICANT {
        int from_table_no;          /* The cube Table In Which
                                       this Implicant is found prime */
        char bin_form[MAX_VARS+1];      /* Binary Form of minterms */
        char to_print_bin[MAX_VARS+1];
        int min_terms_combination[MAX_POW2];
        /* All the minterms which Combined to form this implicant */
        };
struct ALL_IMPLICANTS {
        int no_of_pims;   /* Number of prime implicants */
        int current_position;
        struct IMPLICANT *pm_implicant;
        };


/*      structure of an entry in prime implicant table */

struct PIM_ENTRY {
        int essential;
        int no_of_min_terms;
        int *min_terms_combination;
        int *check_box; /* pointer used to denote check boxes */
        char bin_form[MAX_VARS+1]; /* Binary Form of this implicant */
        char to_print_bin[MAX_VARS+1];
        };
/*      Structure of the prime implicant table  */
struct PIM_TABLE {
        int no_of_entries;  /* No Of implicants in this table */
```

```
struct PIM_ENTRY *pim_entries;
                int *last_row;    /* The last row in the table */
                };
/*      structure of an product */
struct SOP {
        char bin[MAX_VARS+1];      /* Binary Form */
        char pdts[MAX_VARS+1];/* In alphabetical representation */
        struct SOP *next;          /* Pointer to the next product */
};

/*variable Used to hold the first product and number of products */
struct ESS {
        int max;
        struct SOP *begin;
};


/*      Functions In the module sop_pim.obj */
extern void find_prime_implicants();
extern void int_swap(int *x,int *y);


/* Function In  The File sop_main        */
void    build_pim_table(void);
void    display_pim_table(int new,char *message);
void    reverse_pim_table(void);
void    sort_pim_table(void);
void    make_marks_intable(void);
void    reduce_pim_table(void);
int     checkfor_ess_pims(void);
void    note_down_ess_pims(void);
void    status(char *message);
void    pick_all_pims(void);
void    construct_next_pim_table(void);
void    message(char *message);
void    print_sop_list(void);
void    error(char *msseage);
void    form_sop_equation(char *sop_eqn);
void    draw_gates(char *sop_eqn);
/* Function In  The File sop_main        */

/*     In ----sop_funct.c-----    */
int     pow2_of(int x);
void    add_to_sop_list(char *bin);
void    bin_to_chars(char *bin,char *pdts);
void    apply_rule_1(void);
void    apply_rule_2(void);
int     chek_dominance(int x,int y);
void    delete_entries(int max,int *result);
void    delete_entry(int entry_no);
void    copy_entry(int from,int to);
int     chek_same(int x,int y);
void    copy_2_next_table(int to,int from,int no_of_min_terms);
```

```
void    remove_x_pims(void);
int     all_x_terms(int entry_no);

/*    In ----sop_eqn.c-----     */
void form_sop_equation(char *equation);

/*    In ----sop_gate.c----     */
float get_ell_x(float from_y,float centr_x,float centr_y,
float mag_rad,float min_rad);
void draw_not(int length,int height,int x,int y);
void draw_gates(char *equation);
void draw_input_lines(void);
void connect(int x,int y);
void draw_not(int length,int height,int x,int y);
void draw_and(int length,int height,int tlx,int tly,char *eqn);
void opengraph(void);
float to_radius(int ang);
void draw_or(int height,int x,int y);
void connect_and_or(int dif,int x1,int y1,int x2,int y2);

/*    In ----sop_grap.c----     */
void print_in_graphics(void);
int find_int_arr_max(int max,int *arry);
void display_pim_table(int statt,char *mess);
void mark_yes(int x,int y);
void message(char *mess);
void message(char *mess);
void status(char *mess);
void error(char *mess);
int count_cost(char *term);

/*    In ----sop_sort.c----     */
void reverse_pim_table(void);
void int_arr_addres_swap(int **x,int **y);
void sort_pim_table(void);
void swap_pim_tbl_entries(struct PIM_ENTRY *x,struct PIM_ENTRY *y);
int count_ones(char *x);
void int_arr_swap(int entries,int *x,int *y);
```

```
/*                                                                      *
 *This file contains all the structure declarations and function        *
 *      Prototypes.                                                     *
 *                                                                      */
#include "sop_main.h"

/*                      Global Variables.                               */
struct ESS essential;
struct PIM_TABLE pim_table;
struct ALL_IMPLICANTS pims_list;
int max_min_terms;
int max_vars;
int max_pow2;
int MIN_TERMS[MAX_POW2];

main()
{
        int ess_pims;
        char *sop_eqn;
/*                                                                      *
 *To find out prime implicants.  Prime Implicants are stored in the    *
 *      global variable.                                                *
 *                                                                      */
        find_prime_implicants();
/*                                                                      *
 *      Contruct Prime Implicant Table From The Prime Implicants.       *
 *                                                                      */
        build_pim_table();
       display_pim_table(1,
            "Fresh Table Constructed With Prime implicants");

/*                                                                      *
 *To Arrange The Table According To Cost Of Each Prime Implicant.      *
 *                                                                      */
        reverse_pim_table();
        display_pim_table(1,"After Reversing The Table");
        sort_pim_table();
        display_pim_table(1,"After Sorting Table");
/*                                                                      *
 *To Place Checks in The columns coresponding To the minterms           *
 *contained in the prime Implicant listed on that row.                 *
 *                                                                      */
        make_marks_intable();
        display_pim_table(0,"    Marking Min_Terms");
/*                                                                      *
 *      To delete unneccessary prime implicants.                        *
 *                                                                      */
        reduce_pim_table();
        display_pim_table(1,"Reduced Pim_Table");

        while (1)
        {
```

```
/*                                                                    *
 *      To Find out essential prime implicants in the table.          *
 *                                                                    */
                ess_pims = checkfor_ess_pims();
                display_pim_table(0,"Checking For Essenial Pims...");
                if ( ess_pims != 0)
                {
/*                                                                    *
 *              Some prime Implicants are essential. Note It.         *
 *                                                                    */
                        note_down_ess_pims();
                }
                if( ess_pims == ALL )
                {
/*                                                                    *
 *      All The minterms are covered. Goto Draw Circuit.  *
 *                                                                    */
                        status("All Min Terms Are Covered\n");
                        break;
                }
                else
                {
                        if (ess_pims == 0 )
                        {
                                sound(500);
                                delay(100);
                                sound(750);
                                delay(150);
                                nosound();
                                pick_all_pims();
                                break;
                        }
                        else
                        {
/*                                                                    *
 *              Some Minterms are left uncovered so construct next*
 *              table.                                               *
 *                                                                   */
                                status("Constructing Next Table..");
                                construct_next_pim_table();
                                message("The Next Table..");
                                display_pim_table(1,"Next Table");
/*                                                                    *
 * To Place Checks in The columns coresponding To the minterms       *
 *      contained in the prime Implicant listed on that row.         *
 *                                                                   */
                                make_marks_intable();
display_pim_table(0,"Marking Essential Prime Implicants..");
```

```
/*                                                                    *
 *          To delete unneccessary prime implicants.                  *
 *                                                                   */
        reduce_pim_table();
        display_pim_table(1,"Reduced Prime Implicant Table");
                        }
                }
        }

/*      All PRODUCTS are listed                                      */


if ( (sop_eqn = (char *)malloc((essential.max * max_vars +
                essential.max + 1)*sizeof(char)))==NULL)
        {
                error("Can't Find Memory for sop_equation");
        }

/*      To Combine All terms into equation.                          */
        form_sop_equation(sop_eqn);

/*      To Draw The logical diagram.                                 */
        draw_gates(sop_eqn);
        closegraph();
/*      Return To the Operating System.                              */
        return 0;
}

/*      Functions are defined in the following files.                */

#include "sop_grap.c"
#include "sop_sort.c"
#include "sop_ess.c"
#include "sop_func.c"
#include "sop_eqn.c"
#include "sop_gate.c"


/*Forms the final equation by Including All the prime implicants.   */

void pick_all_pims(void)
{
        int i;
        essential.max = 0;
        for(i=0;i<pims_list.no_of_pims;++i)
        {
        add_to_sop_list(pims_list.pm_implicant[i].to_print_bin);
        }
        return;
}
```

```
/*      Standard include files.                                    */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <alloc.h>
#include <graphics.h>

#define MAX_VARS        6   /*    Maximum No Of Variables.*/
#define MAX_POW2        64

/*Constants used to help understanding the program.      */
#define NO              (-111)
#define YES             111

/*Function prototypes Used in the program.               */
void pause(void);
void read_min_terms();
void to_binary(int max,int number,char *bin,char *rev,int *ones);
void sort_first_table();
void sort_array();
void swap();
void construct_0_cube();
void print_cube_table();
void print_cube_table_g();
void build_table0_info();
void print_table_info();
void sort_table();
void cube_swap();
void int_swap();
void str_swap();
void int_arr_copy();
void int_arr_concat();
void construct_next_table();
int  count_entries_in_next_table();
int  is_diff_by_abit(char *x,char *y);
void init_next_info();
void check_for_pm_implicants();
void add_to_pim_list();
void print_pims_list(void);
void find_prime_implicants();

extern error();
extern void opengraph();
extern int pow2_of();
extern int find_int_arr_max(int max,int *arry);


/*Structure Declarations.                                    */
struct TABLE {          /* The First table.                  */
        int min_term_value;
        char bin_form[MAX_VARS+1];
        char to_print_bin[MAX_VARS+1];
        int no_of_ones;
       };
```

```
struct CUBE_TABLE {      /* Cube Table.                              */
        int order;
        int group;
        int min_terms_combination[MAX_POW2];
        char bin_form[MAX_VARS+1];
        char to_print_bin[MAX_VARS+1];
        int is_checked;
        int is_prime_implicant;
        };

struct CUBE_TABLE_INFO {/* This structure is used to hold */
        int order;       /* details about a cube table.   */
        int max_no_of_entries;
        int groups;
        int begin_position[MAX_VARS+1];
        int no_of_entries[MAX_VARS+1];
        };


struct IMPLICANT {       /* A prime implicant                        */
        int from_table_no;
        char bin_form[MAX_VARS+1];
        char to_print_bin[MAX_VARS+1];
        int min_terms_combination[MAX_POW2];
        };
struct ALL_IMPLICANTS { /* All prime implicants.                     */
        int no_of_pims;
        int current_position;
        struct IMPLICANT *pm_implicant;
        };
#define XMAX    640
#define YMAX    320
int sscr_mode;
```

```
#include "sop_pim.h"

/*      External variables defined in sop_main.c        */
extern struct ALL_IMPLICANTS pims_list;
extern int max_vars;
extern int max_pow2;
extern int max_min_terms;
extern int MIN_TERMS[MAX_POW2];

/*      Global  variables.                              */
int no_of_xs;
int no_of_1s;

/*                                                              *
 *      This function coordinates all functions to find out prime  *
 *      implicants out of all implicants.                       *
 *                                                              */
void find_prime_implicants()
{
        int i,j,next_table_entries,next=0;
        struct TABLE *First;
        struct CUBE_TABLE *cube[MAX_VARS+1];
        struct CUBE_TABLE *temp_cube;
        struct CUBE_TABLE_INFO *table_info;

        while(1)
        {
                printf("Enter No Of Variables :");
                scanf("%d",&max_vars);
                if ( max_vars <= 0 )
                        printf("Incorrect Entry..\n");
                if ( max_vars != 0 && max_vars <= MAX_VARS )
                        break;

                if ( max_vars > MAX_VARS)
                printf("The Maximum No Of Variables is %d\n",
                                                        MAX_VARS);
        }
        max_pow2 = pow((float)max_vars,2.0);
        if (( temp_cube = ( struct CUBE_TABLE *)
        malloc(MAX_POW2 * 2 * sizeof(struct CUBE_TABLE)) ) == NULL)
        {
                error("Not Enough Memory");
        }
        do
        {
                printf("Enter Max Min_Terms :");
                scanf("%d",&max_min_terms);
                if ( max_min_terms == 0)
                {
                        error("Impossible To HAve No Min_Terms");
                }
```

```
    if( max_min_terms > max_pow2 )

        {
                printf("%d Variables Can't Have %d Min_Terms\n\
                        Reduce terms\n",max_vars,max_min_terms);
                }
                else
                        break;
        }
        while (1);
        if (( pims_list.pm_implicant = (struct IMPLICANT *)
        malloc(max_min_terms * sizeof(struct IMPLICANT))) == NULL)
        {
                error("Can't Find Enough Memory For Pim List");
        }
        if (( First = (struct TABLE *)
        malloc(max_min_terms * sizeof(struct TABLE)) )==NULL)
        {
        printf("Can't Find memory For First Table\n");
        printf("And So Insufficient Memory. Can't Proceed.\n\
                                Try To Reduce Max_min_Terms\n");
                error("");
        }

/*      Reading all the minterms.                                   */
        read_min_terms(First);

        if ( no_of_1s == max_pow2)
        {
/*      All the combinations have been entered.                     */
                printf("The Final SOP is := 1\n");
                exit(0);
        }
        if( no_of_xs == max_pow2)
        {
/*                                                                  *
        It is not neccessary to go further since all the terms are  *
        don't cares.                                                *
 *                                                                  */
                printf("The Final SOP is := 0\n");
                exit(0);
        }

/*      printf("ConVersion Of Minterm List To BinaRy Form\n");
        printf("=========================================\n\n");
        printf("#    min_term    Binary_form     No_of ones\n");
        for(i=0;i<max_min_terms;++i)
        {
                printf("%2d     M %2d      :=     %*s       := %d\n",
                        i,First[i].min_term_value,MAX_VARS,
                        First[i].to_print_bin,First[i].no_of_ones);
        }
        printf("\n************************************************\
                                        **********************\n");
```

```
        printf("Sorted Minterm List\n");
        printf("=========================================\n\n");
        printf("#     min_term     Binary_form      No_of ones\n");
*/
sort_first_table(First);
/*printf("\n*****************************************************\
                                        *********************\n");
        for(i=0;i<max_min_terms;++i)
        {
                printf("%2d     M %2d      :=     %*s       := %d\n",i,
                First[i].min_term_value,max_vars,
                        First[i].to_print_bin,First[i].no_of_ones);
        }
        printf("\n*************************************************\
                                        ******************\n");
*/
        if (( cube[0] = (struct CUBE_TABLE *)
        malloc( max_min_terms * sizeof(struct CUBE_TABLE)) )==NULL)
        {
                perror("malloc");
                printf("Can't Find Memory For cube[0]\n");
                printf("And So Insufficient Memory. Can't Proceed.\n\
                                Try To Reduce Max_min_Terms\n");
                error("");
        }

/*To put entries into cube table from the first table.  */
        construct_0_cube(First,cube[0]);
        if (( table_info = (struct CUBE_TABLE_INFO *)
        malloc((max_vars+1) * sizeof(struct CUBE_TABLE_INFO)))==NULL)
        {
                perror("malloc");
                printf("Can't Find Memory For table_info\n");
                printf("And So Insufficient Memory. Can't Proceed.\n\
                                Try To Reduce Max_min_Terms\n");
                error("");
        }

/*      Collect informations about cube table.          */
        build_table0_info(cube[0],&table_info[0]);
        print_cube_table_g(cube[0],table_info[0],
"Table just Constructed from minterm");
/* ----
        print_cube_table(cube[0],&table_info[0]);
        printf("Table Informations\n");
        print_table_info(&table_info[0],max_vars);

---- */

/*To arrange the cube table according to cost of minterms.    */
        sort_table(cube[0],&table_info[0],max_vars);
        print_cube_table_g(cube[0],table_info[0],
"After Sorting according to cost");
```

```
/* ----
        printf("Sorted 0-Cube Table\n");
        print_cube_table(cube[0],&table_info[0]);
---- */

        for(i=1;( i < max_vars+1 ) && (next != NO ) ;++i)
        {
/*                                                                      *
 *       Using a temporary cube table construct next table and
return     *
 *      no of entries in it.                                            *
 *                                                                      */
        next_table_entries=count_entries_in_next_table(i,
                cube[i-1],temp_cube,&table_info[i-1],&table_info[i]);
/*      printf("Number Of Entries Coming In the Next (%d) Table is\
                        = %d\n",i,next_table_entries);*/
                if ( next_table_entries == 0 )
                {
/*              All The Possible Cubes Are Formed.                      */
                        next = NO;
/*              Add prime implicants found in the table to list.        */
                        add_to_pim_list(cube[i-1],&table_info[i-1]);
                        break;
                }
        if (( cube[i] = (struct CUBE_TABLE *)
        malloc( next_table_entries * sizeof(struct CUBE_TABLE))
                                                        )==NULL)
                {
                perror("malloc");
                printf("Can't Find Memory For cube[%d]\n",i);
                printf("And So Insufficient Memory. Can't Proceed.\n\
                        Try To Reduce Max_min_Terms\n");
                        error(1);
                }

/*      To construct next cube table.                                   */
                construct_next_table(i,cube[i-1],cube[i],
                        &table_info[i-1],&table_info[i]);
/* ----                                         %d Follws\n",i);
        printf("The New Table Info(%d) \n",i);
        print_table_info(&table_info[i],max_vars);
        printf("The %d Table after Forming %d Table\n",i-1,i);
        print_cube_table(cube[i-1],&table_info[i-1]);
---- */
/                                                                       *
 *      Scan the whole cube table and mark the implicants              *
 *      that are not covered in constructing next table.               *
 *      These are prime implicants from that table.                    *
 *                                                                     */
check_for_pm_implicants(cube[i-1],&table_info[i-1]);
print_cube_table_g(cube[i-1],table_info[i-1],
"After Checking For Prime Implicants");
```

```
        print_cube_table_g(cube[i],table_info[i],"The Next Table");

 /*----
                printf("The Newly Constructed %d Table\n",i);
                print_cube_table(cube[i],&table_info[i]);
---- */
                if ( i == max_vars)
                {
/*      The Maximum Possible Order cube table.                  */
                add_to_pim_list(cube[i],&table_info[i]);
                }
        }
        for(j=0;j<i;++j)
                free(cube[j]);
        free(table_info);
        free(temp_cube);
        free(First);

/*      To print all prime implicants.                          */
        print_pims_list();
        max_min_terms=no_of_1s;
        getch();
        putchar('\n');
        return;

}

/*Functions are defined in this file.*/
#include "sop_sub.c"
#define NEW     (-123)
#define OLD     (-100)

/*                                                                      *
 *This function constructs next cube table from the a cube table       *
 *with the help of informations in pres_info and also records           *
 *next table's info in next_info.                                       *
 *                                                                     */
void construct_next_table(int pres_order,struct CUBE_TABLE *pres_table,
        struct CUBE_TABLE *next_table,
        struct CUBE_TABLE_INFO *pres_info,
                        struct CUBE_TABLE_INFO *next_info)
{
   int ii;
   int i,j,k,order,nxt_tbl_pos=0,pres_order_pow2,pres_order_p2_1,
                                        next_group=0,reduns=0;
   int gp_1_beg_pos,gp_2_beg_pos,gp_1_end_pos,gp_2_end_pos,diff;
   char gp_1_bin[MAX_VARS+1],gp_2_bin[MAX_VARS+1],
        next_bin[MAX_VARS+1],next_to_print_bin[MAX_VARS+1];
   int gp_1_combns[MAX_POW2],gp_2_combns[MAX_POW2],
                                        next_combns[MAX_POW2];

        pres_order_pow2 = (int)pow(2.0,(double)pres_order);
        pres_order_p2_1 = (int)pow(2.0,(double)(pres_order-1));
```

```
/*      Initializes the variable next_info.                    */
        init_next_info(next_info,pres_order,max_vars);

        for(i=0;i<= max_vars;++i)
        {
                if(pres_info->no_of_entries[i] == 0 )
                {
/*No Entries In cube table of order (pres_order-1) for group (i)  */
                        continue;
                }
                else
                {
                gp_1_beg_pos = pres_info->begin_position[i];
                gp_1_end_pos =
                   gp_1_beg_pos + pres_info->no_of_entries[i];

                gp_2_beg_pos = pres_info->begin_position[i+1];

                gp_2_end_pos =
                gp_2_beg_pos + pres_info->no_of_entries[i+1];

                        if(gp_2_beg_pos == NO )
                        {
                                continue;
                        }
                        else
                        {
                          next_group = NEW;
                          for(j=gp_1_beg_pos;j<gp_1_end_pos;++j)
                                {
                                order = pres_info->order;
                                next_info->order = order+1;
                                strcpy(gp_1_bin,
                                        pres_table[j].bin_form);

                                for(k=gp_2_beg_pos;k<gp_2_end_pos;
                                                        ++k)
                                        {
KK:
strcpy(gp_2_bin,pres_table[k].bin_form);
diff = is_diff_by_abit(gp_1_bin, gp_2_bin);
if(diff == NO)
{
        continue;
}
else
{
        if( next_group==NEW)
        {
                next_info->begin_position[i] = nxt_tbl_pos;
                next_group = OLD;
        }
```

```
        pres_table[j].is_checked = YES;
        pres_table[k].is_checked = YES;
    strcpy(next_table[nxt_tbl_pos].bin_form,pres_table[j].bin_form);
        next_table[nxt_tbl_pos].bin_form[diff] ='x';
        for(ii=0;ii<nxt_tbl_pos;++ii)
        {
        if (strcmp(next_table[ii].bin_form,
             next_table[nxt_tbl_pos].bin_form)==0)
                {
/*       Redundancy Comes Between (ii) and nxt_tbl_pos.          */
                        k++;
                        reduns++;
                        goto KK;
                }
        }
        ++(next_info->no_of_entries[i]);
        next_table[nxt_tbl_pos].order = pres_order;
        next_table[nxt_tbl_pos].group = i;
int_arr_copy(pres_order_p2_1,pres_table[j].min_terms_combination,
                                        next_combns);
int_arr_concat(pres_order_p2_1,next_combns,
                        pres_table[k].min_terms_combination);

int_arr_copy(pres_order_pow2,next_combns,
        next_table[nxt_tbl_pos].min_terms_combination);

strcpy(next_table[nxt_tbl_pos].to_print_bin,
                        pres_table[j].to_print_bin);

next_table[nxt_tbl_pos].to_print_bin[max_vars-1-diff] = 'x';
next_table[nxt_tbl_pos].is_checked = NO;
next_table[nxt_tbl_pos].is_prime_implicant = NO;
nxt_tbl_pos++;
}
                                        }
                                }
        if ( next_group == NEW )
                {
                next_info ->no_of_entries[i] = 0;
                next_info -> begin_position[i] = NO;
/*      No Matches Found Between group(i) and group(i+1).  *
 *      In cube Table Of order pres_order-1.               */
                                }
                        }
                }
        }
        next_info->max_no_of_entries = nxt_tbl_pos;
        return;
}

/*                                                                  *
 *This function constructs next table as above and returns no of    *
 *entries in it.                                                    *
 *                                                                  */
```

```
int count_entries_in_next_table(int pres_order,
          struct CUBE_TABLE *pres_table,
   struct CUBE_TABLE *next_table,struct CUBE_TABLE_INFO *pres_info,
                                  struct CUBE_TABLE_INFO *next_info)
{
 int ii;
 int i,j,k,order,nxt_tbl_pos=0,pres_order_pow2,next_group=0,reduns=0;
 int gp_1_beg_pos,gp_2_beg_pos,gp_1_end_pos,gp_2_end_pos,diff;
 char gp_1_bin[MAX_VARS+1],gp_2_bin[MAX_VARS+1],next_bin[MAX_VARS+1],
                                  next_to_print_bin[MAX_VARS+1];
int gp_1_combns[MAX_POW2],gp_2_combns[MAX_POW2],
                                  next_combns[MAX_POW2];

        pres_order_pow2 = (int)pow(2.0,(double)pres_order);
        init_next_info(next_info,pres_order,MAX_VARS);
        for(i=0;i<= max_vars;++i)
        {
                if(pres_info->no_of_entries[i] == 0 )
                {
/*No Entries In cube table of order(pres_order-1) for group (i)*/
                        continue;
                }
                else
                {
                 gp_1_beg_pos = pres_info->begin_position[i];
                 gp_1_end_pos =
                   gp_1_beg_pos + pres_info->no_of_entries[i];

                 gp_2_beg_pos = pres_info->begin_position[i+1];

                 gp_2_end_pos =
                   gp_2_beg_pos + pres_info->no_of_entries[i+1];

                        if(gp_2_beg_pos == NO )
                        {
/*              No Entries In The Next Lower Group.             */
                                /* i++; */
                                continue;
                        }
        else
        {
                next_group = NEW;
                for(j=gp_1_beg_pos;j<gp_1_end_pos;++j)
                {
                        order = pres_info->order;
                        next_info->order = order+1;
                        strcpy(gp_1_bin,
                                pres_table[j].bin_form);

                        for(k=gp_2_beg_pos;k<gp_2_end_pos:
                                                ++k)
                        {
KK:
strcpy(gp_2_bin,pres_table[k].bin_form);
```

```
diff = is_diff_by_abit(gp_1_bin,gp_2_bin);
if(diff == NO)
{
        continue;
}
else
{
        if( next_group == NEW )
        {
                next_info->begin_position[i] = nxt_tbl_pos;
                next_group = OLD;
        }
 pres_table[j].is_checked = YES;
 pres_table[k].is_checked = YES;
 strcpy(next_table[nxt_tbl_pos].bin_form,pres_table[j].bin_form);
 next_table[nxt_tbl_pos].bin_form[diff] ='x';
 for(ii=0;ii<nxt_tbl_pos;++ii)
        {
         if (strcmp(next_table[ii].bin_form,
                        next_table[nxt_tbl_pos].bin_form)==0)

                {
/*Redundancy Comes Between (ii) and (nxt_tbl_pos).       */
                        ++k;
                        reduns++;
                        goto KK;
                }
        }

   ++(next_info->no_of_entries[i]);
   next_table[nxt_tbl_pos].order = pres_order;
   next_table[nxt_tbl_pos].group = i;
   int_arr_copy(pres_order,pres_table[j].min_terms_combination,
                                                   next_combns);
        int_arr_concat(pres_order,next_combns,
                        pres_table[k].min_terms_combination);

        int_arr_copy(pres_order_pow2,next_combns,
                next_table[nxt_tbl_pos].min_terms_combination);

        strcpy(next_table[nxt_tbl_pos].to_print_bin,
                                pres_table[j].to_print_bin);
        next_table[nxt_tbl_pos].to_print_bin[max_vars-1-diff] = 'x';
        next_table[nxt_tbl_pos].is_checked = NO;
        next_table[nxt_tbl_pos].is_prime_implicant = NO;
        nxt_tbl_pos++;
}
                                        }
                                }
        if ( next_group == NEW )
                {
                next_info ->no_of_entries[i] = 0;
                next_info -> begin_position[i] = NO;
```

```
/*      No Matches Found Between group(i) and group(i+1)                *
 *      In cube Table Of order (pres_order-1).                          */
                }
                        }
                }
        }
        next_info->max_no_of_entries = nxt_tbl_pos;
        return (nxt_tbl_pos);
}

/*                                                                      *
 *This function checks all the entries in a cube table and marks        *
 *all prime implicants.                                                 *
 *                                                                      */
void check_for_pm_implicants(struct CUBE_TABLE *cube,
                        struct CUBE_TABLE_INFO *table_info)
{
        static pims_count=0,current;
        int i,j,order_pow2;

        order_pow2 = (int)pow(2.0,(double)table_info->order);
        current = pims_list.current_position;
        for(i=0;i< table_info->max_no_of_entries;++i)
        {
                if(cube[i].is_checked == NO )
                {
/*An Entry In Table (table_info->order) Is Found UnChecked. */

pims_list.pm_implicant[current].from_table_no =
                                table_info->order;

        strcpy(pims_list.pm_implicant[current].bin_form,
                                        cube[i].bin_form);

        strcpy(pims_list.pm_implicant[current].to_print_bin,
                                        cube[i].to_print_bin);

        int_arr_copy(order_pow2,
                cube[i].min_terms_combination,
                        pims_list.pm_implicant[current].
                        min_terms_combination);
                        pims_list.no_of_pims++;
                        pims_list.current_position++;
                        pims_count++;
                        current++;
                }
        }
        return;
}

/*                                                                      *
 *This function adds all the prime implicants that are in a cube       *
 *to the prime implicants list.                                         *
 *                                                                      */
```

```
void add_to_pim_list(struct CUBE_TABLE *cube,
                        struct CUBE_TABLE_INFO *table_info)
{
        int i,j,order_pow2,current;

        order_pow2 = (int)pow(2.0,(double)table_info->order);
        current = pims_list.current_position;
        for(i=0;i< table_info->max_no_of_entries;++i)
        {
                if(cube[i].is_checked == NO )
                {
/*This entry (i) is a prime implicant.                  */
pims_list.pm_implicant[current].from_table_no =
                                        table_info->order;
strcpy(pims_list.pm_implicant[current].bin_form,
                                    cube[i].bin_form);

strcpy(pims_list.pm_implicant[current].to_print_bin,
                        cube[i].to_print_bin);

        int_arr_copy(order_pow2,
                cube[i].min_terms_combination,
                pims_list.pm_implicant[current].
                        min_terms_combination);

        pims_list.no_of_pims++;
        pims_list.current_position++;
                current++;
                }
        }
        return;
}

/*      This function prints all prime implicants in the list.*/
void print_pims_list()
{
        int i,j,order_pow2,order;
        char tmp_str[100],tmps[100];

        cleardevice();
sprintf(tmp_str,"No Of Prime Implicants := %d",pims_list.no_of_pims);
        rectangle(0,0,XMAX-2,YMAX-2);
        rectangle(2,2,XMAX-4,YMAX-4);
        outtextxy(20,10,tmp_str);
        for(i=0;i<pims_list.no_of_pims;++i)
        {
                order = pims_list.pm_implicant[i].from_table_no;
                order_pow2 = (int)pow(2.0,(double)order);
                strcpy(tmp_str,"");
                for(j=0;j<order_pow2;++j)
                {
                        if(j==0)
                sprintf(tmps,"%d",pims_list.pm_implicant[i].
                                min_terms_combination[j]);
```

```
                    else
                    sprintf(tmps,",%d",pims_list.pm_implicant[i].
                                        min_terms_combination[j]);
                            strcat(tmp_str,tmps);
                    }
outtextxy(25,20+i*12,tmp_str);
outtextxy(25+XMAX/2,20+i*12,pims_list.pm_implicant[i].to_print_bin);
          }
}
#include "sop_ctb.c"
```

```
/*      To  print  the   cube  table.                                   */
void print_cube_table_g(struct CUBE_TABLE *cube,
                struct CUBE_TABLE_INFO table_info,char *mesg)
{
 int pow_2,tabl_length,tabl_height,x_start,y_start,pr_imp=0,grp=0;
 int max_mint,dig_max,x_chars,fir,secon,x_dist,y_dist,i,j;
char tmp_str[80],tmps[10];

        if(sscr_mode == 0)
        {
                opengraph();
                sscr_mode = 1;
        }
        cleardevice();
        pow_2 = pow2_of(table_info.order);
        max_mint = find_int_arr_max(max_min_terms,MIN_TERMS);
        sprintf(tmp_str,"%d",max_mint);
        dig_max = strlen(tmp_str);
        x_chars = 3+2+(pow_2*dig_max)+pow_2-1+2+max_vars;
        tabl_length=x_chars*9;
        if(tabl_length>XMAX)
        {
                outtextxy(10,20,
          "Can't Print table Screen Length is not sufficient");
                return;
        }
        x_start = (XMAX-tabl_length)/2;
        tabl_height=(table_info.max_no_of_entries+1)*12;
        if(tabl_height>YMAX)
        {
                outtextxy(10,20,
         "Can't Print table Screen height is not sufficient");
                return;
        }
        y_start = (YMAX-tabl_height)/2;
        outtextxy(10,20,mesg);
        rectangle(x_start,y_start,XMAX-x_start,YMAX-y_start);
        fir = x_start+3*9;
        secon=x_start+(x_chars-2-max_vars)*9;
        y_dist=tabl_height/(table_info.max_no_of_entries+1);
        x_dist=tabl_length/x_chars;
        line(fir,YMAX-y_start,fir,y_start+y_dist);
        line(secon,YMAX-y_start,secon,y_start+y_dist);
        line(x_start,y_start+y_dist,XMAX-x_start,y_start+y_dist);
        sprintf(tmp_str,"%d-Cubes",table_info.order);
        outtextxy(fir,y_start+3,tmp_str);
        for(i=0;i<table_info.max_no_of_entries;++i)
        {
                if(table_info.no_of_entries[grp]==pr_imp)
                {
                pr_imp=0;
                grp++;
                line(x_start,y_start+y_dist*(i+1),XMAX-x_start,
                                        y_start+y_dist*(i+1) );
```

```
                }
                pr_imp++;
                tmp_str[0]='\0';
                for(j=0;j<pow_2;++j)
                {
                        if(j==0)
                                sprintf(tmps,"%d",
                                 cube[i].min_terms_combination[j]);
                        else
                                sprintf(tmps,",%d",
                                 cube[i].min_terms_combination[j]);
                        strcat(tmp_str,tmps);
                }
                outtextxy(x_start+5,y_start+3+(i+1)*y_dist,
                        (cube[i].is_checked==YES?"{":"*"));
                outtextxy(secon+5,y_start+3+(i+1)*y_dist,
                                cube[i].to_print_bin);
                outtextxy(fir+5,y_start+3+(i+1)*y_dist,tmp_str);
        }
        getch();
        return;
}
```

```
/*This function reads all the minterms.                          */
void read_min_terms(struct TABLE *First)
{
        int i,j,tt;

        while(1)
        {
                printf("Enter No Of One Terms: ");
                scanf("%d",&no_of_1s);
                if ( no_of_1s > max_min_terms)
                {
        printf("Impossible To Have %d terms since you have\
 set max_min_terms = %d \n",no_of_1s,max_min_terms);
                }
                if ( no_of_1s == 0 )
                {
                printf("Incorrect Use Of Quine-McCluskey Method.\
                       Sorry!!!!\n");
                        exit(1);
                }
                if( no_of_1s <= max_min_terms)
                        break;
        }
        no_of_xs = max_min_terms - no_of_1s;

        for(i=0;i<max_min_terms;++i)
        {
        AA:
            if ( i < no_of_1s )
                printf("%3d..Enter One Min_Term#%d :",i,i);
           else
                printf("%3d..Enter Don't Care Min_Term#%d :",i,
                                        i-no_of_1s);
                scanf("%d",&(First[i].min_term_value));
        if ( First[i].min_term_value >= (int)pow(2.0,max_vars))
        {
        printf("Impossible Minterm value (%d) [Maximum =%d ]\n",
                First[i].min_term_value,(int)pow(2.0,max_vars)-1);
                goto AA;
        }
        for(j=0;j<i;++j)
        {
        if ( First[i].min_term_value ==
                                First[j].min_term_value )
                {
                BB:     printf("This Term Is Already In List.\n");
                        printf("0.May I Ignore This Term\n\t\tor\n");
                        printf("1.Do You Want To Enter All Terms \
                                        From Beginning?\n");
                        printf("                                                [0/1]:");
                        scanf("%d",&tt);
                          if(tt == 0 )
                           {
                                goto AA;
```

```
                                }
                            else if ( tt == 1 )
                                {
                                   i = 0;
                                goto AA;
                                    }
                        else goto BB;
                        }
                }
        to_binary(max_vars,First[i].min_term_value,
                                First[i].bin_form,
                                First[i].to_print_bin,
                                &(First[i].no_of_ones));
        }

/*Storing all the one terms in the global int array MIN_TERMS.*/
           for(i=0;i<no_of_1s;++i)
                 MIN_TERMS[i] = First[i].min_term_value;
/*Sorting it in ascending  order*/
           sort_array(MIN_TERMS,no_of_1s);
           return;
        }

/*                                                                    *
 *This function sorts an integer array of max elements in            *
 *       ascending order.                                             *
 *                                                                    */
  void sort_array(int *MIN,int max)
  {
         int i,j;
         for(i=0;i<max-1;++i)
           {
            for(j=i;j<max;++j)
              {
                 if ( MIN[i] > MIN[j] )
                    int_swap(&MIN[i],&MIN[j]);
              }
           }
         return;
  }
/*                                                                    *
 *This function converts an integer to binary form and stores it     *
 * a character array. Also counts number of ones in it.              *
 *                                                                    */
  void to_binary(int max,int number,char *bin,char *rev,int *ones)
  {
         int i,bit;
         i=0;
         for(i=0;i<max;++i)
           {
              rev[i]=bin[i]='0';
           }
         i=0;
         *ones = 0;
```

```
        while(number>0 && i < max)
            {
                bit = number % 2;
                if ( bit == 1 )
                   (*ones)++;
                rev[max-i-1] = bin[i] = bit + '0';
                number = number / 2;
                i++;
            }
        bin[max] = '\0';
        rev[max] = '\0';
        return;
    }
/*                                                                    *
 *This function sorts the first table in ascending order according*
 *       to no_of_ones in the minterms.                               *
 *                                                                    */
  void sort_first_table(struct TABLE *first)
  {
        int i,j;
        for(i=0;i<max_min_terms-1;++i)
          {
             for(j=i+1;j<max_min_terms;++j)
               {
                  if( first[i].no_of_ones > first[j].no_of_ones)
                    {
                        swap(&first[i],&first[j]);
                    }
               }
          }
  }
/*                                                                    *
 *     This function interchanges two entries of type TABLE          *
 *       in the first table.                                          *
 *                                                                    */
    void swap(struct  TABLE *x,struct TABLE *y)
    {
       int_swap(&(x->min_term_value),&(y->min_term_value));
       int_swap(&(x->no_of_ones),&(y->no_of_ones));
       str_swap(x->bin_form,y->bin_form);
       str_swap(x->to_print_bin,y->to_print_bin);
    }
/*                                                                    *
 *  This function builds the first cube table i.e) cube table of*
    order zero, by transfering all entries from the first table.*
 *                                                                    */
void construct_0_cube(struct TABLE *First,struct CUBE_TABLE *cube_0)
{
     int i,j;
     for(i=0;i<max_min_terms;++i)
       {
        cube_0[i].order = 0;
        cube_0[i].group = First[i].no_of_ones;
        cube_0[i].min_terms_combination[0] = First[i].min_term_value;
```

```
            strcpy(cube_0[i].bin_form,First[i].bin_form);
            strcpy(cube_0[i].to_print_bin,First[i].to_print_bin);
            cube_0[i].is_checked = NO;
            cube_0[i].is_prime_implicant = NO;
         }
            return;
   }

/*       This function prints the cube table.                          */
void print_cube_table(struct CUBE_TABLE *cube,
       struct CUBE_TABLE_INFO *info)
{
      int max_combination,i,j,max_entries;
      max_entries = max_min_terms;
      max_entries = info->max_no_of_entries;
      max_combination = (int)pow(2.0,(double)cube[0].order);
      printf("#  ones  Min_Term  Bin_Form  Checked Prime_Implicant\n");
      for(i=0; i < max_entries ;++i)
        {
           printf("%d    %d   ",i,cube[i].group);
           for(j=0;j<max_combination;++j)
             printf("%2d ",cube[i].min_terms_combination[j]);
             printf("%*s ",max_vars,cube[i].to_print_bin);
            printf(" %13s ",(cube[i].is_checked==YES ) ? "YES" : "NO");
            printf(" %13s ",
                  (cube[i].is_prime_implicant == YES )? "YES" : "NO");
              printf("\n");
         }
         return;
}

/*                                                                      *
 *   This function records the position of implicants according        *
 *   to their no_of_ones count.                                        *
 *                                                                     */
void build_table0_info(struct CUBE_TABLE *cube,
                       struct CUBE_TABLE_INFO *info)
{
        int i;
        info->order = cube[0].order;
        info->max_no_of_entries = max_min_terms;
        info->groups = MAX_VARS + 1;
        for(i=0;i<max_vars+1;++i)
        {
                info->no_of_entries[i]= 0;
                info->begin_position[i] = NO;
        }
        for(i=0;i<max_min_terms;++i)
        {
           if(info->no_of_entries[cube[i].group] == 0 )
                {
                 info->begin_position[cube[i].group] = i;
                }
                ++(info->no_of_entries[cube[i].group]);
```

```
        }
        return;
}
/*                                                                      *
*  This function prints all the informations about a cube table        *
*  recorded in info.                                                   *
*                                                                      */
void print_table_info(struct CUBE_TABLE_INFO *info,int max_groups)
{
        int i;
   printf("Maximum No Of Entries in cube table %d = %d\n",
          info->order,info->max_no_of_entries);
   printf("No Of Groups In This Table is = %d\n",info->groups);
   printf("Group   Begin_Pos      No_Of_Entries\n");
    for(i=0;i<=max_groups;++i)
      {
        printf("%4d %d  %5d        %6d\n",info->order,i,
                info->begin_position[i],info->no_of_entries[i]);
        }
        return;
}

/*                                                                      *
 * This function arranges the cube table by sorting the implicants     *
 *       in all the groups individually.                                *
 *                                                                      */
void sort_table(struct CUBE_TABLE *cube,
                struct CUBE_TABLE_INFO *table_info,int max_vars)
{
        int i,j,k,start,end,order;
        order = 0;
        for(i=0;i<max_vars+1;++i)
        {
            if ( table_info->no_of_entries == 0 )
                {
                  continue;
                }
                else
                {
                  start = table_info->begin_position[i];
                  end = start + table_info->no_of_entries[i];
                   if ( end > max_min_terms)
                     {
                        printf("Table Entry Error!!\n");
                        printf("Table Info Says A Position \
                            Which Goes beyond Max_min_Terms\n");
                                exit();
                        }
                        for(j=start;j<end-1;++j)
                        {
                         for(k=j+1;k<end;++k)
                           {
          if ( cube[j].min_terms_combination[order] >
                        cube[k].min_terms_combination[order])
```

```
                {
                        cube_swap(&cube[j],&cube[k]);
                        }
                        }
                        }
                }
        }
        return;
}

/*    This function interchanges two entries of type CUBE_TABLE.    */
void cube_swap(struct CUBE_TABLE *x,struct CUBE_TABLE *y)
{
        int i,max_combination;
        max_combination = (int)pow(2.0,(double)x->order);
        int_swap(&(x->order),&(y->order));
        int_swap(&(x->group),&(y->group));
        for(i=0;i<max_combination;++i)
        int_swap(&(x->min_terms_combination[i]),
                        &(y->min_terms_combination[i]));
        str_swap(x->bin_form,y->bin_form);
        str_swap(x->to_print_bin,y->to_print_bin);
        return;
}

/* This function interchanges two integers.      */
void int_swap(int *x,int *y)
{
        int t;
        t = *x;
        *x = *y;
        *y = t;
}

/*This function interchanges two strings.        */
void str_swap(char *x,char *y)
{
        char temp[MAX_VARS+1];
        strcpy(temp,x);
        strcpy(x,y);
        strcpy(y,temp);
}

/*                                                                      *
 *This function copies max entries from an integer array to another *
 *integer array.                                                       *
 *                                                                     */
void int_arr_copy(int max,int *from,int *to)
{
        int i;
        for(i=0;i<max;++i)
        {
                to[i] = from[i];
        }
```

```
}

/*                                                                      *
 * This function concatinates an integer array with another array       *
 * after max entries.                                                   *
 *                                                                      */
void int_arr_concat(int max,int *x,int *y)
{
        int i,tmp;
        for(i=0;i<max;++i)
        {
                x[max+i] = y[i];
        }
}

/*                                                                      *
 * This function returns YES if two strings differ at only one         *
 *  position.                                                           *
 *                                                                      */
int is_diff_by_abit(char *x,char *y)
{
        int diff,pos,i=0;
        diff = 0;
        while ( x[i] != '\0')
        {
                if (x[i] == 'x' )
                {
                        if (y[i] != 'x')
                        {
/*              X's Doesn't Match!!                             */
                                return NO;
                        }
                }
                if ( x[i] != y[i] )
                {
                        diff++;
                        if ( diff == 1)
                        {
                                pos = i;
                        }
                }
        i++;
        }
        if ( diff == 1 )
                return pos;
        else
                return NO;
}

/*  This function sets all the members in info to default values.*/
void init_next_info(struct CUBE_TABLE_INFO *info,
                    int order,int max_groups)
{
        int i;
```

```
        info->max_no_of_entries = 0;
        info->order = order;
        info->groups = max_vars+1;
        for(i=0; i <= max_groups;++i)
        {
                info->begin_position[i] = NO;
                info->no_of_entries[i] = 0;
        }
        return;
}
```

```
#define XMAX     640
#define YMAX     320
#define YLEAVE   10
#define XLEAVE   20

/*      Global Variables                        */
int max_tms;    /*row and column positions of the table.          */
int terms_xpos,*row_pos,*col_pos;
/*maximum no of minterms in an impicant and the positions of each */

int row_length,col_height;/* Length of a row and Height of a column*/

/*      Screen Mode                             */
int scr_mode;
/*                                                                      *
 *This function calculates the positions of vertical and horizontal    *
 *lines for the table to be printed.                                    *
 *                                                                      */
void print_in_graphics(void)
{
        int *tmp_arr;
        int no_of_entries,i,j,starty,max_terms,
                                        no_of_col,startx;

        no_of_entries = pim_table.no_of_entries;
 if(tmp_arr = (int *)malloc(no_of_entries*sizeof(int))) == NULL)
        {
                error("Can't Find Memory For Tmp_arr");
        }


        for(i=0;i<no_of_entries;++i)
        {
                tmp_arr[i] = pim_table.pim_entries[i].no_of_min_terms;
        }
        max_terms =  find_int_arr_max(no_of_entries,tmp_arr);
        max_tms = max_terms;
        col_height = (YMAX - 2*YLEAVE) /(no_of_entries + 2);
 if ((row_pos = (int *)malloc((no_of_entries+2)*sizeof(int))) == NULL)
        {
                error("Can't Find Memory For Tmp_arr");
        }
        starty = YLEAVE;
        for(i=0;i<no_of_entries+2;++i)
        {
                row_pos[i] = starty;
                starty += col_height;
        }
        no_of_col = max_min_terms + max_terms + 1;
 if ( (col_pos = (int *)malloc(max_min_terms*sizeof(int))) == NULL)
        {
                error("Can't Find Memory For Col_Pos");
        }
        row_length = (XMAX-2*XLEAVE)/(no_of_col);
```

```
        startx = XMAX-XLEAVE-row_length;
        for(j=max_min_terms-1;j>=0;j--)
        {
                col_pos[j]=startx;
                startx -= row_length;
        }
        startx -= row_length*(max_terms-1);
        terms_xpos = startx;
}

/*                                                                      *
 *This  function  finds  out  the  maximum  of an  integer array      *
 *of known size.                                                       *
 *                                                                     */
int find_int_arr_max(int max,int *arry)
{
        int i,j,mm=0;

        mm = arry[0];
        for(i=0;i<max-1;++i)
        {
                if ( mm < arry[i] )
                        mm = arry[i];
                for(j=i;j<max;++j)
                {
                        if ( mm < arry[j] )
                                mm = arry[j];
                }
        }
        return mm;
}

/*This function displays the prime implicant table.                    */
void display_pim_table(int statt,char *mess)
{
        int i,j,no_of_col,no_of_entries;
        char tmp_str[BUFSIZ],ttmp[BUFSIZ];

        no_of_entries = pim_table.no_of_entries;
        if ( statt == 1)
        {
/*      Table Has Been Modified. So Calculate Positions */
                print_in_graphics();
                cleardevice();
        }
        message(mess);
        rectangle(XLEAVE,YLEAVE,XMAX-XLEAVE,YMAX-YLEAVE);

        for(i=0;i<no_of_entries+2;++i)
        {
                line(XLEAVE,row_pos[i],XMAX-XLEAVE.row_pos[i]);
        }
        no_of_col = max_min_terms + max_tms + 1;
        i=no_of_col-1;
```

```
for(j=0;j<max_min_terms;j++)
{
        line(col_pos[j],YLEAVE,col_pos[j],YMAX-YLEAVE);
}
line(terms_xpos,YLEAVE,terms_xpos,YMAX-YLEAVE);

for(i=0;i<max_min_terms;++i)
{
        sprintf(tmp_str,"%d",MIN_TERMS[i]);
if ( textwidth(tmp_str) < row_length-row_length/3 )
                outtextxy(col_pos[i]+row_length/3,
                row_pos[0]+col_height/2,tmp_str);
}
for(i=0;i<pim_table.no_of_entries;++i)
{
        if( pim_table.pim_entries[i].essential != NO )
        {
                outtextxy(XLEAVE+row_length/2,
                        row_pos[i+1]+col_height/2,"*");
        }
for(j=0;j<pim_table.pim_entries[i].no_of_min_terms;++j)
        {
                if ( j==0)
                {
                        sprintf(tmp_str,"%d",
                        pim_table.pim_entries[i].
                        min_terms_combination[j]);
                }
                else
                {
                        sprintf(ttmp,",%d",
                        pim_table.pim_entries[i].
                        min_terms_combination[j]);
                        strcat(tmp_str,ttmp);
                }
        }
        if ( textwidth(tmp_str)
                 <
                (col_pos[0]-(terms_xpos+row_length/2)) )
        {
                outtextxy(terms_xpos+row_length/2,
                        row_pos[i+1]+col_height/2,tmp_str);
        }
        for(j=0;j<max_min_terms;++j)
        {
        if ( pim_table.pim_entries[i].check_box[j] != NO )
                {
                mark_yes(col_pos[j],row_pos[i+1]);
                }
        }
}
```

```
 for(i=0;i<max_min_terms;++i)
        {
                if ( pim_table.last_row[i] != NO )
                {
                        mark_yes(col_pos[i],
                        row_pos[pim_table.no_of_entries+1]);
                }
        }
        getch();
        return;
}

/*This function draws a  at a given point.                              */
void mark_yes(int x,int y)
{
        float x1,y1,x2,y2,x3,y3;

        x1 = x+(float)row_length/4.0;
        x2= x+(float)row_length*3.0/8.0;
        x3 = x+(float)row_length*7.0/8.0;

        y1 = y+(float)col_height/2.0;
        y2 = y + (float)col_height*5.0/8.0;
        y3 = y+(float)col_height/4.0;
        line(x1,y1,x2,y2);
        line(x2,y2,x3,y3);
        return;
}

/*This function displays a string in the graphics screen.              */
void message(char *mess)
{
        int i[] = { 0,0,XMAX,0,XMAX,10,0,10,0,0},j,tmp;

        setfillstyle(SOLID_FILL,0);
        tmp = getcolor();
        setcolor(getbkcolor());
        fillpoly(4,i);
        setcolor(tmp);
        outtextxy(0,0,mess);
        return;
}

/*                                                                      *
 *This function displays a string in the lower position of             *
 *the screen in graphics screen.                                       *
 *                                                                     */
void status(char *mess)
{
int i[] = { XLEAVE,YMAX-10,XMAX,YMAX-10,XMAX,YMAX,XLEAVE,YMAX,
        XLEAVE,YMAX-10},tmp;

        setfillstyle(SOLID_FILL,0);
        tmp = getcolor();
```

```
        setcolor(getbkcolor());
        fillpoly(4,i);
        setcolor(tmp);
        outtextxy(XLEAVE,YMAX-10,mess);
        return;
}

/*This function displays the error message and aborts the program. */
void error(char *mess)
{
        char tmp[BUFSIZ];

        if ( scr_mode == 0 )
        {
                printf("Error: %s\n",mess);
                printf("\07Press Any Key..");
                getch();
                putchar('\n');
        }
        else
        {
                sprintf(tmp,"Error: %s",mess);
                outtextxy(10,100,tmp);
                getch();
                closegraph();
        }
        exit();
        return;

/* This function counts number of 1s in a string.*/
int count_cost(char *term)
{
        int i=0,count=0;
        while(term[i] != '\0')
        {
                if (term[i] == '1' )
                        count ++;
                i++;
        }
        return count;
}
```

```
/*This Function Reverses the prime implicant table.                    */

void reverse_pim_table(void)
{
        int i,max;

        max = pim_table.no_of_entries;
        for(i=0;i<max/2;++i)
        {
int_swap(&pim_table.pim_entries[i].no_of_min_terms,
        &pim_table.pim_entries[max-i-1].no_of_min_terms);

int_swap(&pim_table.pim_entries[i].essential,
        &pim_table.pim_entries[max-i-1].essential);

int_arr_addres_swap(&pim_table.pim_entries[i].min_terms_combination,
        &pim_table.pim_entries[max-i-1].min_terms_combination);

str_swap(pim_table.pim_entries[i].bin_form,
        pim_table.pim_entries[max-i-1].bin_form);

str_swap(pim_table.pim_entries[i].to_print_bin,
        pim_table.pim_entries[max-i-1].to_print_bin);
        }
}

/*
 *  This function interchanges two pointers of type pointer to          *
 *  array of integers.                                                   *
 *                                                                      *
void int_arr_addres_swap(int **x,int **y)                               */
{
        int *t;

         t = *x;
        *x = *y;
        *y =  t;

        return;
}

/*
 *This function arranges the prime implicant table according to        *
 *number of Xs in the implicant.                                       *
 *                                                                     *
void sort_pim_table(void)                                              */
{
        int beg_pos[MAX_VARS+1],entries[MAX_VARS+1];
        int i,j,k,max_nos,group=0,gp_1_start,gp_1_endd;

        for(i=0;i<max_vars+1;++i)
        {
                beg_pos[i] = NO;
                entries[i] = 0;
```

```
        }
        max_nos = pim_table.pim_entries[0].no_of_min_terms;
        beg_pos[0] = 0;
        for(i=0;i < pim_table.no_of_entries; ++i)
        {
        if(pim_table.pim_entries[i].no_of_min_terms == max_nos)
         {
                entries[group]++;
                }
                else
                {
                        group++;
                        beg_pos[group]=i;
                        entries[group]++;
                }
                max_nos = pim_table.pim_entries[i].no_of_min_terms;
        }

        for(i=0;i<max_vars+1;++i)
        {
                if ( entries[i] == 0 )
                {
/*      No Entries In This Group.                                       */
                        continue;
                }
                gp_1_start = beg_pos[i];
                gp_1_endd  = beg_pos[i] + entries[i];
                for(j=gp_1_start;j<gp_1_endd-1;++j)
                {
                for(k=j+1;k<gp_1_endd;++k)
                {
                if(count_ones(pim_table.pim_entries[j].
                                                bin_form)
                                >
                        count_ones(pim_table.pim_entries[k].
                                        bin_form) )
                        {
                        swap_pim_tbl_entries(
                                &pim_table.pim_entries[j],
                                &pim_table.pim_entries[k]);
                        }
                        }
                }
        }

}

/*This function interchanges two implicants in the table. */

void swap_pim_tbl_entries(struct PIM_ENTRY *x,struct PIM_ENTRY *y)
{
        int x_no_of_terms,y_no_of_terms;

        x_no_of_terms = x->no_of_min_terms;
```

```
        y_no_of_terms = y->no_of_min_terms;

        if( x_no_of_terms != y_no_of_terms)
        {
                x_no_of_terms,y_no_of_terms);
        }
    int_swap(&x->essential,& y->essential);
    int_arr_swap(x_no_of_terms,x->min_terms_combination,
                        y->min_terms_combination);

   str_swap(x->bin_form,y->bin_form);
   str_swap(x->to_print_bin,y->to_print_bin);
}

/*This Function returns number of 1s in the string.        */
int count_ones(char *x)
{
        int i=0,ones=0;

        while(x[i] != '\0')
        {
                if ( x[i] == '1' )
                        ones++;
                i++;
        }
        return ones;
}

/*This function Swaps all the entries of two integer arrays.      */
void int_arr_swap(int entries,int *x,int *y)
{
        int i;

        for(i=0;i<entries;++i)
        {
                int_swap(&x[i],&y[i]);
        }
}
```

```
/*
 * This function inspects for columns containing only a single mark  *
 * to find out essential prime implicant and if found implicant's    *
 * essential member is set to 1.                                     *
 * Returns no of essential prime implicants contained in this table. *
 * If None found returns NO.                                         *
 * If all the minterms are covered returns ALL.                     */
 *
int checkfor_ess_pims()
{
        int i,j,k,entry_no,ess,col,ess_entries=0,all;

        for(i=0;i<max_min_terms;++i)
        {
            ess = YES;
            for(j=0;j<pim_table.no_of_entries;++j)
             {
               if (  pim_table.pim_entries[j].check_box[i] == YES )
                 {
                        entry_no = j;
                        break;
                 }
              }
                 if ( j == pim_table.no_of_entries )
                        continue;
                 for(j=0;j<pim_table.no_of_entries;++j)
                 {
                    if( j == entry_no )
                    continue;
                     if ( pim_table.pim_entries[j].check_box[i]==YES)
                          {
                                ess = NO;
                                break;
                          }
                 }
                 if ( ( ess != NO) && pim_table.last_row[i] == NO )
                 {
                   ess_entries++;
                   pim_table.pim_entries[entry_no].essential = 1;
                   for(j=0;j<pim_table.pim_entries[entry_no].
                                        no_of_min_terms;++j)
                         {
                           col = min_term_2_col_pos(pim_table.
                         pim_entries[entry_no].min_terms_combination
                                                          [j]);
                                pim_table.last_row[col] = YES;
                         }
                 }
        }
        for(i=0;i<max_min_terms;++i)
        {
                if ( pim_table.last_row[i] == YES )
                {
                        all = YES;
```

```
                }
                else
                {
                        all = NO;
                        break;
                }
        }
        if ( all == YES )
        {
/*       All Min_Terms Are Covered.                                    */
                return ALL;
        }
        all = NO;
        for(i=0;i<pim_table.no_of_entries;++i)
        {
                if ( pim_table.pim_entries[i].essential == NO )
                {
                        all = NO;
                        break;
                }
                else
                        all = YES;
        }
        if (all == YES )
        {
/*        All The Entries Are Essential.                               */
                return ALL;
        }
        if( ess_entries == 0 )
                return 0;       /* Go To Enumeration Step */
        return ess_entries;
}

/*                                                                      *
 * This Function places marks in the check boxes corresponding to *
 * the minterms contained in the prime implicant.                       *
 *                                                                      */
void make_marks_intable()
{
        int i,j,col;

        for(i=0;i<pim_table.no_of_entries;++i)
        {
          for(j=0;j<pim_table.pim_entries[i].no_of_min_terms;++j)
           {
                col=min_term_2_col_pos(pim_table.pim_entries[i].
                                        min_terms_combination[j]);
                        if ( col != NO )
                           pim_table.pim_entries[i].check_box[col]
                                                        = YES;
           }
        }
}
```

```
/*
 * This function returns the position of a minterm in the *
 * array MIN_TERMS.                                        *
 *                                                         *
int min_term_2_col_pos(int term)                           */
{
        int i;

        for(i=0;i<max_min_terms;++i)
        {
                if ( MIN_TERMS[i] == term )
                        return i;
        }
        return NO;
}
```

```
/* This function calculates 2 power x.                              */
int pow2_of(int x)
{
        return ((int)pow(2.0,(double)x));
}

/*
 * This function transfers all the prime implicants in the          *
 * structure ALL_IMPLICANTS to PIM_TABLE.                           *
 * The first prime implicant table is constructed in this function.*
 *                                                                  */
void build_pim_table()
{
        int i,j,pow2;

        pim_table.no_of_entries = pims_list.no_of_pims;
        if ( (pim_table.pim_entries = (struct PIM_ENTRY *)
           malloc(pims_list.no_of_pims * sizeof(struct PIM_ENTRY)))
                                                        == NULL)
        {
                error("Can't Find Memory To Build PIM_Table");
        }

        for(i=0;i<pims_list.no_of_pims;++i)
        {
          pim_table.pim_entries[i].essential = NO;
          pow2 = pow2_of(pims_list.pm_implicant[i].from_table_no);
          pim_table.pim_entries[i].no_of_min_terms = pow2;
          if( (pim_table.pim_entries[i].min_terms_combination =
                        (int *)malloc(pow2 * sizeof(int))) == NULL)

           {
             error("Can't Find Memory For min_terms_combination");
           }
                if( (pim_table.pim_entries[i].check_box =
                        (int *)malloc(max_min_terms
                         * sizeof(int))) == NULL)
                {
                        error("Can't Find Memory For check_box");
                }
                for(j=0;j<max_min_terms;++j)
                        pim_table.pim_entries[i].check_box[j] = NO;
                for(j=0;j<pow2;++j)
                {
                pim_table.pim_entries[i].min_terms_combination[j] =
                pims_list.pm_implicant[i].min_terms_combination[j];
                }
                strcpy(pim_table.pim_entries[i].bin_form,
                          pims_list.pm_implicant[i].bin_form);
                strcpy(pim_table.pim_entries[i].to_print_bin,
                          pims_list.pm_implicant[i].to_print_bin);

        }
        if((pim_table.last_row = (int *)
```

```
                malloc(max_min_terms * sizeof(int)))==NULL)
        {
           error("Can't Find Memory For pim_table.last_row");
        }
        for(i=0;i<max_min_terms;++i)
                pim_table.last_row[i] = NO;
}

/* This function prints the prime implicant table in stdout     */
void print_pim_table()
{
        int i,j;

        printf("No Of PImplicants In This Table := %d\n",
                                        pim_table.no_of_entries);
        for(i=0;i<pim_table.no_of_entries;++i)
        {
          printf("%s ",pim_table.pim_entries[i].to_print_bin);
          if( pim_table.pim_entries[i].essential == NO )
                {
                        printf("-");
                }
                else
                {
                  for(j=0;j<pim_table.pim_entries[i].essential;++j)
                        {
                                printf("*");
                        }
                }
                printf("    ");
             for(j=0;j<pim_table.pim_entries[i].no_of_min_terms;++j)
                {
                  printf("%*d ",3,pim_table.pim_entries[i].
                                min_terms_combination[j]);
                }
                for(;j<6;++j)
                        printf("    ");
                for(j=0;j<max_min_terms;++j)
                {
                  printf("%s ",(pim_table.pim_entries[i].check_box[j]
                                        == NO ) ? "NO " : "YES");
                }
                printf("\n");
         }
         for(i=0;i<max_min_terms;++i)
         {
         printf("%s ",(pim_table.last_row[i] ==NO ) ? "NO " : "YES");
         }
         printf("\nPress Any Key...\n");
         getch();
 }
```

```
/*                                                                      *
 * This function adds all the essential implicants in the table to      *
 * sop_list.                                                            *
 *                                                                      */
void note_down_ess_pims()
{
        int i;

        for(i=0;i<pim_table.no_of_entries;++i)
        {
                if ( pim_table.pim_entries[i].essential != NO )
                {
                  add_to_sop_list(pim_table.pim_entries[i].
                                                  to_print_bin);
                }
        }
}

/* This function adds a string to the SOP linked list .     */
void add_to_sop_list(char *bin)
{
        struct SOP *pres,*temp;

    if ( (temp  = (struct SOP *)malloc(sizeof(struct SOP)) ) == NULL)
        {
                error("Can't Find Memory For temp_sop_entry");
        }

        temp->next = NULL;
        strcpy(temp->bin,bin);
        bin_to_chars(bin,temp->pdts);
        if( essential.max == 0 )
        {
/*      This is the First entry in the list.                        */
                essential.begin = temp;
                essential.max++;
                return;
        }
        else
                pres = essential.begin;

        while ( pres->next != NULL )
        {
                pres = pres->next;
        }
        pres->next = temp;
        essential.max++;
}

/* This function prints all the products added in the linked list. */
void print_sop_list()
{
        struct SOP *pres;
```

```
        printf("No Of Sops := %d\n",essential.max);
        pres = essential.begin;
        do
        {
                printf("%s  := %s\n",pres->bin,pres->pdts);
                pres = pres->next;
        }
        while ( pres != NULL);
}
/*                                                                      *
 * This function converts binary string to alphabetic string.          *
 * Notation used : Upper case letters for active high - A for A         *
 *                 Lower case letters for active low. - a for A         *
 *                                                                      */
void bin_to_chars(char *bin,char *ch)
{
        int i,j=0;

        for(i=0;i<max_vars&& bin[i] != '\0';++i)
        {
                switch (bin[i] )
                {
                case '1':
                        ch[j] =  'A' + i;
                        j++;
                        break;
                case '0' :
                        ch[j] = 'a' + i;
                        j++;
                        break;
                }
        }
        ch[j] = '\0';
}
/*                                                                      *
 * This function delete unnessary prime implicants in the prime-       *
 * implicant  table.                                                    *
 *                                                                      */
void reduce_pim_table()
{
/*                                                                      *
 * To  remove all the prime implicants that are  formed  by            *
 * combining only don't care minterms.                                  *
 *                                                                      */
   remove_x_pims();
   display_pim_table(1,"After Removing Allx prime implicants");

/*    To delete interchangeable implicants.                         */
        apply_rule_1();
        display_pim_table(1,"After Rule#1");

/*  To delete implicants that are dominated by other implicant.     */
        apply_rule_2();
        display_pim_table(1,"After Rule#2");
```

```
        return;
}

/*                                                                      *
 *This function finds out all the rows that cover the same minterms    *
 *and deletes one which costs more than the other.                      *
 *                                                                      */
void apply_rule_1()
{
   int orig_max,i,j,same=0,*result,res_tbl_count=0;
   orig_max = pim_table.no_of_entries;
    if ( (result = (int *)malloc(orig_max * sizeof(int))) == NULL)
        {
                error("CAn't Find Memory In apply_rule_1");
        }
        for(i=0;i<orig_max;++i)
                result[i] = NO;

        for(i=orig_max-1;(i>=0) && (result[i] == NO);--i)
        {
                for(j=i-1;j>=0;--j)
                {
                        same=chek_same(i,j);
                        if(same==YES)
                        {
                                result[res_tbl_count] = i;
                                res_tbl_count++;
                        }
                }
        }

          if ( res_tbl_count != 0 )
                  delete_entries(res_tbl_count,result);
          return;
}

/*                                                                      *
 *This function finds out all the implicants that are dominated by     *
 *some other implicant and removes it from the prime implicant table    *
 *                                                                      */
void apply_rule_2()
{
  int orig_max,i,j,same=0,*result,res_tbl_count=0;

  orig_max = pim_table.no_of_entries;
   if ( (result = (int *)malloc(orig_max * sizeof(int))) == NULL)
        {
                error("CAn't Find Memory In apply_rule_1");
        }
        for(i=0;i<orig_max;++i)
                result[i] = NO;
        for(i=0; (i<orig_max) ;++i)
        {
                if ( result[i] != NO )
```

```
                        continue;
                for(j=i+1;j<orig_max;++j)
                {
                        same=chek_dominance(i,j);
                        if(same==YES)
                        {
                                result[res_tbl_count] = j;
                                res_tbl_count++;
                        }
                }
        }
          if ( res_tbl_count != 0 )
                  delete_entries(res_tbl_count,result);
          return;
}

/*                                                                      *
 *   This function compares two prime implicants and finds out if      *
 *   any one the implicant is dominated by the other.                  *
 *                                                                     */
int chek_dominance(int x,int y) /* weather x Dominates y */
{
        int i;

        for(i=0;i<max_min_terms;++i)
        {
                if(pim_table.pim_entries[y].check_box[i] == YES )
                {
                  if ( pim_table.pim_entries[x].check_box[i] != YES )
                        return NO;
                }
        }
        return YES;
}


/*                                                                      *
 * This function deletes all the implicants whose positions are        *
 * given in an array from the table.                                    *
 *                                                                     */
void delete_entries(int max,int *result)
{
        int i;

        for(i=0;i<max;++i)
        {
                delete_entry(result[i]);
        }
        pim_table.no_of_entries -= max;
        return;
}
```

```
/*
 *     This function deletes one implicant from the                    *
 *     prime implicant table.                                          *
 *                                                                     *
void delete_entry(int entry_no)                                        */
{
        int i,orig;

        orig = pim_table.no_of_entries;
        for(i=entry_no;i<orig-1;++i)
        {
                copy_entry(i+1,i);
        }
        return;
}

/*
 *This function copies the whole implicant entry from one position    *
 *another position in the prime implicant table.                       *
 *                                                                     *
void copy_entry(int from,int to)                                       */
{
        pim_table.pim_entries[to].essential =
                        pim_table.pim_entries[from].essential;

        pim_table.pim_entries[to].no_of_min_terms =
                        pim_table.pim_entries[from].no_of_min_terms;

        int_arr_copy(pim_table.pim_entries[from].no_of_min_terms,
           pim_table.pim_entries[from].min_terms_combination,
                pim_table.pim_entries[to].min_terms_combination);

        int_arr_copy(max_min_terms,
                pim_table.pim_entries[from].check_box,
                        pim_table.pim_entries[to].check_box);

        strcpy(pim_table.pim_entries[to].bin_form,
                        pim_table.pim_entries[from].bin_form);

        strcpy(pim_table.pim_entries[to].to_print_bin,
                        pim_table.pim_entries[from].to_print_bin);
        return;
}


/*
 * This function compares two prime implicants and finds out both     *
 * are covering the same minterms.                                     *
 *                                                                     *
int chek_same(int x,int y)                                             */
{
        int i;

        for(i=0;i<max_min_terms;++i)
```

```
        {
          if(pim_table.pim_entries[x].check_box[i] == YES )
                {
                 if ( pim_table.pim_entries[y].check_box[i] != YES )
                                return NO;
                }
                if(pim_table.pim_entries[y].check_box[i] == YES)
                {
                  if(pim_table.pim_entries[x].check_box[i] != YES )
                                return NO;
                }
        }
        return YES;
}

/*                                                                      *
 * This function builds the next prime implicant table from the        *
 * previous table. It acts upon the uncovered minterms.                 *
 *                                                                     */
void construct_next_pim_table()
{
   int *new_MIN_TERMS;
   int i,j,nxt_tbl_pos=0,no_of_new_min_terms=0,new_min_count=0;
   int no_of_old_entries;

        no_of_old_entries = pim_table.no_of_entries;
        for(i=0;i<max_min_terms;++i)
        {
                if (pim_table.last_row[i] == NO )
                        new_min_count++;
        }
        if ( (new_MIN_TERMS = (int *)
                malloc(new_min_count * sizeof(int))) == NULL)
        {
                error("Can't Find Memory For New_MIn_TERS[i]");
        }
        for(i=0,j=0;i<max_min_terms;++i)
        {
                if ( pim_table.last_row[i] == NO )
                {
/*                      Row(i) is Found Unchecked                       */
                        new_MIN_TERMS[j++]=MIN_TERMS[i];
                }
        }
        no_of_new_min_terms = j;
        for(i=0;i<no_of_old_entries;++i)
        {
                for(j=0;j<max_min_terms;++j)
                {
                        if( (pim_table.last_row[j] == NO)
                                        &&
                   (pim_table.pim_entries[i].check_box[j] == YES ) )

                        {
```

```
/*      Entry No(i) Is To Be Included In Next Table.                    */
                                copy_2_next_table(nxt_tbl_pos,i,
                                        no_of_new_min_terms);
                                nxt_tbl_pos++;
                                break;
                        }
                }
        }

        pim_table.no_of_entries = nxt_tbl_pos;
        max_min_terms = no_of_new_min_terms;
        for(i=0;i<no_of_new_min_terms;++i)
        {
                MIN_TERMS[i] = new_MIN_TERMS[i];
                pim_table.last_row[i] = NO;
        }
        return;
}

/*                                                                      *
 * This function copies one prime implicant into the next table        *
 *      and initializes all the members.                                *
 *                                                                      */
void copy_2_next_table(int to,int from,int no_of_min_terms)
{
        int i;

        pim_table.pim_entries[to].essential = NO;
        pim_table.pim_entries[to].no_of_min_terms =
                pim_table.pim_entries[from].no_of_min_terms;
        int_arr_copy(pim_table.pim_entries[from].no_of_min_terms,
                pim_table.pim_entries[from].min_terms_combination,
                pim_table.pim_entries[to].min_terms_combination);

        strcpy(pim_table.pim_entries[to].bin_form,
                        pim_table.pim_entries[from].bin_form);

        strcpy(pim_table.pim_entries[to].to_print_bin,
                pim_table.pim_entries[from].to_print_bin);

        for(i=0;i<no_of_min_terms;++i)
                pim_table.pim_entries[to].check_box[i] = NO;
        return;
}

/*                                                                      *
 * This function searches the prime implicant table and finds out       *
 *all the implicants that are formed by combining only don't care      *
 *   minterms. Finally deletes all this entries from the table.        *
 *                                                                      */
void remove_x_pims()
{
        int i,res_tbl_count=0,*result;
```

```
if ( (result = (int *)malloc(pim_table.no_of_entries * sizeof(int)))
                        == NULL)
        {
                error("Can't Find Memory In remove-x_pims");
        }
        for(i=0;i<pim_table.no_of_entries;++i)
        {
                if(all_x_terms(i)==YES)
                {
                        result[res_tbl_count] = i;
                        res_tbl_count++;
                }
        }
        delete_entries(res_tbl_count,result);
        return;
}

/*                                                                      *
 *  This function searches an implicnat's minterms combination and     *
 *       finds out whether they are all don't care minterms.            *
 *                                                                      */
int all_x_terms(int entry_no)
{
        int i;

        for(i=0;i<max_min_terms;++i)
        {
        if(pim_table.pim_entries[entry_no].check_box[i] != NO )
                        return NO;
        }
        return YES;
}
```

```
 /*This function combines all the products into a single  string.  */

void form_sop_equation(char *equation)
{
        struct SOP *pres;

        pres = essential.begin;
        strcpy(equation,pres->pdts);
        pres  = pres->next;
        while ( pres != NULL )
        {
                strcat(equation," + ");
                strcat(equation,pres->pdts);
                pres = pres->next;
        }
        return;
}
```

```
/*    Constants used in the graphics part of the program. */

#define XMAX    640     /* The resolution of the screen */
#define YMAX    320

#define GRAPH_LIB "\\tc\\bgi"   /* The Graphics Library          */
#define XSTART 10
#define YSTART  5
#define GAP 15   /* Gap between two adjacent input lines.        */


/*      Global variables Used by all functions in this part.     */
int *inputs;              /* X axis Position of the input lines.  */
int no_of_and_gates;      /* Number of and gates required.        */
int *and_endx,*and_endy;/* Coordinates of AND gates' positions. */

/*                                                                *
 *Thin function draws the circuit for an equation. All the products *
 *are taken from the global variable essential(The SOP linked list).*
 *                                                                */
void draw_gates(char *equation)
{
        int i,Inity,level1x,height,no_of_gates;
        char tmp[80];
        struct SOP *pres;

        cleardevice();
        rectangle(0,0,XMAX-1,YMAX-1);
        no_of_gates=0;
        pres = essential.begin;
        while ( pres != NULL )
        {
                no_of_gates++;
                pres = pres->next;
        }
        inputs = (int *)malloc(2*max_vars*sizeof(int));
        and_endx = (int*)malloc(no_of_gates*sizeof(int));
        and_endy = (int*)malloc(no_of_gates*sizeof(int));
        for(i=0;i<2*max_vars;++i)
        inputs[i] = XSTART + i * GAP;
        draw_input_lines();
        level1x = XSTART + GAP * 2 * max_vars + 2 *GAP;
        height = 25;

        height = (YMAX - YSTART - (YSTART+75-4-(GAP/2)))/no_of_gates;
        if ( height > 75 )
                height = 75;
        Inity = YSTART+75-4-GAP-GAP/2-10 + GAP +2;
        pres = essential.begin;
        for(i=0;i<no_of_gates;++i)
        {
        draw_and(height,height-10,level1x,Inity+height*i,pres->pdts);
                pres = pres->next;
```

```
        }
        draw_or(150,380,and_endy[no_of_and_gates/2]-150/2);
        strcpy(tmp," Y = ");
        strcat(tmp,equation);
        outtextxy(XMAX-textwidth(tmp)-10,YMAX-10,tmp);
        getch();
        return;
}

/*                                                                      *
 *      This function draws all the input lines.                        *
 *                                                                      */
void draw_input_lines(void)
{
        int i;
        char chr='A',ch[5];

        for(i=0;i<2*max_vars;++i)
        {
          line(inputs[i],i%2==0?YSTART:YSTART+75-GAP/2-10-2,
                                                  inputs[i],YMAX);
                if ( i % 2 == 0 )
                {
                        sprintf(ch,"%c",chr++);
                        outtextxy(inputs[i] - 10 ,YSTART,ch);
                }

                if ( i %2 == 1 )
                {
                        draw_not(GAP*2-GAP*0.3,GAP,inputs[i],
                                        YSTART+75-4-GAP-GAP/2-10);
                 line(inputs[i-1],YSTART+75-4-2*GAP-10,inputs[i],
                                                YSTART+75-4-2*GAP-10);
                   line(inputs[i],YSTART+75-4-2*GAP-10,inputs[i],
                                                YSTART+75-4-GAP-GAP/2-10);
                       connect(inputs[i-1],YSTART+75-4-2*GAP-10);
                }
        }
        return;
}

/*                                                                      *
 *This function fills the sorroundings of a point to show connection.
 *                                                                      */
void connect(int x,int y)
{
        fillellipse(x,y,2,2);
        return;
}

/*                                                                      *
 *This function draws a NOT gate at a given point with given length    *
 *      and height.                                                     *
 *                                                                      */
```

```
void draw_not(int length,int height,int x,int y)
{
moveto(x-length/2,y);
linerel(length,0);
linerel(-length/2,height);
lineto(x-length/2,y);
fillellipse(x,y+height,(int)((double)height * 0.2),
(int)((double)height * 0.15));
}

/*                                                                    *
 * This function draws an AND gate a point with given length and     *
 *      height then connects all the inputs given in a string.       *
 *                                                                   */
void draw_and(int length,int height,int tlx,int tly,char *eqn)
{
        int grac,start,end,i=0,no_of_inp,pos;
        int step;

        grac = (double)height * 0.15;
        start = tly + grac;
        end = tly + height - grac;
        while(eqn[i]!='\0')
                i++;
        no_of_inp=i;
        if ( no_of_inp == 1)
        {
                if ( eqn[0] >= 'a' && eqn[0] <= 'z' )
                        pos = 2*(eqn[0]-'a')+1;
                if ( eqn[0] >='A' && eqn[0] <= 'Z' )
                        pos = 2 * (eqn[0]-'A');
                line(inputs[pos],start+height/2,tlx+length+length/2,
                                                start+height/2);
                connect(inputs[pos],start+height/2);
                and_endx[no_of_and_gates] = tlx+length+length/2;
                and_endy[no_of_and_gates] = tly+height/2;
                no_of_and_gates++;
                return;
        }
        step = ( end - start ) / (no_of_inp-1);
        moveto(tlx,tly);
        lineto(tlx,tly+height);
        linerel(length,0);
        moveto(tlx,tly);
        linerel(length,0);
        ellipse(tlx+length,tly+height/2,270,90,length/2,height/2);
        and_endx[no_of_and_gates] = tlx+length+length/2;
        and_endy[no_of_and_gates] = tly+height/2;
        no_of_and_gates++;

        for(i=0;i<no_of_inp;++i)
        {
                if ( eqn[i] >= 'a' && eqn[i] <= 'z' )
                        pos = 2*(eqn[i]-'a')+1;
```

```
                if ( eqn[i] >='A' && eqn[i] <= 'Z' )
                        pos = 2 * (eqn[i]-'A');
                line(inputs[pos],start+step*i,tlx,start+step*i);
                connect(inputs[pos],start+step*i);
        }

        return;
}

/*                                                                      *
 *      This function initializes the screen to graphics mode.          *
 *                                                                      */
void opengraph(void)
{
        int gd=DETECT,gm,gerr;
        initgraph(&gd,&gm,GRAPH_LIB);
        gerr = graphresult();
        if ( gerr  != grOk )
        {
                printf("Graphics Error: %s\n",grapherrormsg(gerr));
                printf("Press Any Key ....");
                getch();
                putchar('\n');
                exit();
        }
        scr_mode=1;
        return;
}

/*                                                                      *
 *      This function converts an angle in degrees to radians.          *
 *                                                                      */
float to_radius(int ang)
{
        return (M_PI * (double) ang /180.0);
}

/*                                                                      *
*This Function draws the OR gate at a given point with given height.
 *      Establishes connection from all AND gates.                      *
 *                                                                      */
void draw_or(int height,int x,int y)
{
        int length = 100.0 * height / 70.0,no_of_gates;
        int y1,i,ystep,xstep,grac,endx,endy,stx;

        ellipse(x,y+height/2,270,90,length/3,height/2);
        ellipse(x,y+height/2,270,90,length,height/2);
        ystep = height/no_of_and_gates;
        grac = ystep/2;
        endy = y+grac;
        endx = x+length/3;
        if ( no_of_and_gates /2 != 0)step =
                        (x-and_endx[0])/(no_of_and_gates/2);
```

```
        else
         xstep = length/3;
         stx = xstep ;
         for(i=0;i<no_of_and_gates;++i)
         {
         endx = get_ell_x(endy,x,y+height/2,length/3,height/2);
         connect_and_or(stx,and_endx[i],and_endy[i],endx,endy);
                 if ( i < no_of_and_gates/ 2)
                         stx = stx + xstep;
                 else if ( i == no_of_and_gates/2 )
                         stx = stx-xstep;
                         else
                                 stx = stx - xstep;
                 endy = endy + ystep;

         }
         line(x+length,y+height/2,x+length+20,y+height/2);
         connect(x+length+20,y+height/2);
         outtextxy(x+length+15,y+height/2+5,"Y");
         return;
}

/*                                                                      *
 *    This function calculates the X coordinate of an ellipse          *
 *    for a given  Y coordinate value.                                 *
 *                                                                     */
float get_ell_x(float from_y,float centr_x,float centr_y,
                  float mag_rad,float min_rad)
{
         float tmp,res;
         float y;

         y = centr_y - from_y;
         tmp = 1.0- (pow(y,2.0)/pow(min_rad,2.0));
         tmp = pow(mag_rad,2.0)*tmp;
         res = pow(tmp,0.5);
         res = centr_x + res;
         return res;
}

/* This function connects two points using three line segents.      */
void connect_and_or(int dif,int x1,int y1,int x2,int y2)
{
         line(x1,y1,(x2-dif<x1)?x1:x2-dif,y1);
         line((x2-dif<x1)?x1:x2-dif,y1,(x2-dif<x1)?x1:x2-dif,y2);
         line( (x2-dif<x1)?x1:x2-dif,y2,x2,y2);
         return;
}
```

ABOUT 'C' LANGUAGE

'C' is a general purpose structured language. 'C' is characterised by the ability to write very concise source programs, due in part to the number of operators included within the language. It has a relatively small instruction set, though actual implementations include extensive library functions which enhance the basic instruction.

'C' compilers are commonly available for computers of all sizes. The compilers are usually compact, and they generate object programs that are small and highly efficient when compared with programs compiled from other high-level languages.

'C' mainly deal with the same set of objects that the most computers do, namely characters, numbers and addresses. They may be combined and moved about with the usual arthmetic and logical operartions implemented by actual machines.

'C' is a small language, it is a structured language and it can be used to construct programs that execute very rapidly. It has small number of statements and a compact set of functions. Another important characteristic of 'C' is that its programs are highly portable.

'C' For System Software Development:

In 'C' the fundamental data objects are characters, integers of several sizes and floating point numbers. In addition there is a hierarcy of derived data types created with pointer, arrays, structures, unions and functions.

'C' provides the fundamental flow control constructions required for well-structured programs. Statement, grouping, decision making, looping with the termination test at the top or at the bottom and selecting one of a set of possible case.

'C' provides pointer and the ability to do address arithmetic. Any function may be called recursively and its local variables are typically automatic or created a new with each invocation. Since the language 'C' speeds up both development and execution time, we choose 'C'language, for our project to be done.

Features Of 'C' Language:

The following are the main features of 'C' language used in our project.

POINTERS:

A pointer is a variable that represents the location of a data item, such as a variable, or an array element. Pointrs can be used to pass information back and forth between a function and its reference point. Pointers

provide a way to return multiple data items from a function via function arguments. Pointers also permit references to other functions to be specified as arguments to a given function. This has the effect of passing functions as arguments.

Pointers are also associated with arrays and therefore provide an alternate way to access individual array elements. Moreover, pointers provide a convienient way to represent multi-dimensional arrays, allowing a single multi-dimensional array to be replaced by a lower-dimensional array of pointers. This feature permits a collection of strings to be represented within a single array, even though the individual strings may differ in length.

STRUCTURES:

Structure is a data structure whose individual elements can differ in type. Thus a single structure might contain integer elements, floating-point elements and character elements. The individual structure elements are referred to as memebers. The individual members can be ordinary variables, pointers, arrays or other structures of this type. The member names within a particular structure must be distinct from one another, though a member name can be the same as the name of the variable defined outside the structure.

CLOSEGRAPH:

This function is called when we have finished using the Turbo C graphics routines. This function releases the memory allocated by the graphics system and restores the screen to the video mode that was active before initgraph() was called.

ELLIPSE:

This function draws an elliptical arc on the graphics screen, using the current drawing color and line style.

FILLPOLY:

This function draws a polygon using the current drawing color and line style and then fills the polygon using the current fill color and pattern.

INITGRAPH:

You should call this function before using any of the graphics display functions. This function:

* loads approriate graphics driver from a disk file.
* places the system in the requested graphics mode.
* Sets all graphics settings to their default values.

LINE:

This function draws a line in graphics mode using the current line style and thickness and the current drawing color.

LINEREL:

This function draws a line in graphics mode from the current point to a point that is specified distance away.

LINETO:

This function draws a line in graphics mode from the current point to the point specified by the function.

MOVETO:

This function moves the current graphics point to the position specified by the function.

OUTTEXTXY:

This function displays text at the point specified by the parameters in the functon and does not move the current graphics point.

RECTANGLE:

This function draws a rectangle on the screen using the current drawing color and line style and thickness.The First two parameters of the function specify the coordinates of one corner of the rectangle, and the second two parameters give the position of the opposite corner.

SETCOLOR:

This function sets the current drawing color to the value given by color.