SYNTHESIS OF CELLULAR LOGIC USING A BOOLEAN ANALYZER

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

AND RESEARCH IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

BY

NAOMI RAZ

DEPARTMENT OF COMPUTING SCIENCE

UNIVERSITY OF MANITOBA

WINNIPEG, MANITOBA

NOVEMBER 1969

A B S T R A C T

A synthesis procedure for combinational networks using two-input, one-output logical cells is proposed and investigated. This procedure makes extensive use of the Boolean Analyzer (Simulator) which is capable of solving Boolean equations.

The proposed synthesis method starts by considering the output of the combinational network and iterates towards the signal inputs. A system of Boolean equations is solved at every logical level.

In this thesis, the basic iteration procedure (which is to be repeated for every logical level of the network) was programmed. Experiments with this program were carried out and are reported here. They display the versatility and flexibility of the procedure.

A C K N O W L E D G E M E N T

I wish to thank Professor R.D. Connor of the University of Manitoba and Professor G. W. Farnell of McGill University, whose cooperation made possible the writing of this thesis.

I am in debt to my supervisor, Dr. M. A. Marin, Assistant Professor, McGill University for his guidance and advice throughout the course of this work.

Special thanks to Mr. P. Wardle for a review of the manuscript and for his comments and to Mrs. E. Freitag for typing the text of this thesis.

The assistance of the staff members of the Computer Centre of McGill University is very much appreciated.

Most of all, thanks to my husband Uzí, for his encouragement and support throughout the preparation of this thesis.

TABLE OF CONTENTS

I N T R O D U C T I O N

The classical methods for designing switching circuits are based on the theory of switching by means of devices such as relays or transistors.

The developments of today's technology offer more sophisticated and reliable switching devices, such as Integrated Circuits (IC) and their Large Scale Integration (L.S.I.). This new trend in the development of technology forces many changes in switching circuit design techniques. The discrete component design procedures are abandoned in favour of a new technique which considers the switching device as cells with a given logical structure.

The designer of logical networks, who uses large scale integration, is not interested in the internal electronic structure of the cells. All that is necessary for him is the knowledge of the logical function of the cell. Some methods for the design of such systems exist, such as the ones proposed by Minnick (8) and Weiss (5). Most of these methods assume a certain logical arrangement of the cells such as trees or arrays.

In this thesis, a new general approach to the problem is suggested. It is based on the theory of decomposition which was first proposed by Ashenhurst (1) and further developed by Curtis (2). This theory is quite complex when applied to problems with a large number of variables ($N > 4$). Moreover, computer algorithms based on the theory of decomposition use considerable computer time. To overcome this difficulty, A. Svoboda (14) developed the Boolean Analyzer (B.A.).

The B.A., which is a hardware unit attached to a digital computer, can process a large number of Boolean terms in parallel. An investigation of the field of

problems for this B.A. was conducted by Marin (6) who suggested the possibility of using it in the cellular logic synthesis. This thesis explores ways of solving this problem by forming a set of Boolean equations which represent the system's desired output and restrictions. This set of functions (equations) is solved by the B.A.

The B.A. is not yet available and therefore a Boolean Analyzer Simulator (B.A.S.) was produced. The first version of B.A.S. was developed by Marin (6) for the UCLA Sigma 7 computer system. The IBM 360/75 Fortran IV version of B.A.S. is developed in this thesis. It is an extended version of the sigma 7 one with the advantage of not containing any special assemble routines or dimensional restrictions.

In addition, a computer program is presented here. The program implements the algorithms to find a desired cell configuration to create a certain output function. This function is to be synthesized while being subject to the restrictions of the system as required by the designer. This program uses the B.A.S. and is flexible enough to accommodate a large number of variables and a variety of limitations of the system.

Fortran IV was chosen as most computer installations have a Fortran IV compiler and thus could use the developed program. The memory size of the computer used is the limiting factor of the number of variables. The examples which will be discussed in this thesis demonstrate the use of this tool in the synthesis of combinational logic networks that use only five different cells, constituting a complete set of functions.

Special attention is given to one particular example which realizes functions of four variables with a two-level logic, using the given set of cells. Other examples demonstrate the flexibility of the program in regard to

the number of variables and types of restrictions.

The results obtained from these examples proved one limitation of the proposed method. Because of the combinational nature of these problems, the B.A. would be used quite frequently. Using the B.A.S. instead makes the procedure slow and costly. If a hardware B.A. had been used, and also a hardware unit for the "decomposition" ("DECOMP") process, the method would prove to be faster in finding acceptable solutions and thus more economical and practical.

C H A P T E R O N E

1.1 DEFINITION OF THE PROBLEM

The logic circuits which are being increasingly utilized in science and industry are getting bigger, more complex and covering a larger number of variables. Because of the more frequent use of this type of network, the devices from which they are constructed have changed in a relatively short time from Relays through tubes and transistors to Integrated Circuits (I.C.'s). Because of the use of I.C.'s and the large number of variables involved, new synthesis methods are required.

In general, the problem is to find a method to realize logic networks of the following nature: they yield at their output a desired function of as many variables as necessary. The input, in the form of input variables, is any data presented in binary form, including the output of other similar networks. They are built with I.C.'s or building blocks or cells. In particular we shall here consider only networks made up of 2-input and 1-output cells, realizing only one logic function. These limitations are imposed here to simplify the presentation and the examples.

Let us consider the 16 logic functions of 2 Boolean variables. Out of these, five can be chosen in such a way that they constitute a complete set (see description by Minnick (8) and Weiss (15). This fact will reduce the required number of cell types to five, each one implementing one function out of the chosen complete set.

It would be desirable to achieve an optimum network for the realisation of every particular function, but, since the optimization criteria are different in each application (for example, they could be represented by requirements such as: minimum logic levels or minimum number of cells or the use of only one kind of cell, etc.) this aspect is not to be discussed in this thesis, though some provisions are made to enable optimization as well (Section 2.4).

1.2 EXISTING METHODS OF SOLUTION

Most of the methods for synthesizing switching circuits with cells are based on a cascade configuration. The first to use such a form was K. K. Maitra (5). In his work (published in 1960) he studies switching networks which were constructed by cascading 2-input and 1-output binary logical cells. He assumed that each cell could realise any one of the sixteen possible switching functions of the two binary variables (inputs) and that the inputs to the network appear at the cell terminals in a prescribed order. An n-input "Maitra Cascade" is composed of (n-1) two-input single-output logical cells. The inputs to a typical cell consist of a signal variable (input to the network) and the output from the preceding cell (Fig. 1). Maitra (5) describes a method for enumerating the class of switching functions realisable by cascaded networks and the conditions for realizing arbitrary switching functions by such networks. He presents a map method for synthesizing such networks. A later work by J. Skalansky (12) proposes a synthesis procedure for the same type of cascades which does not specify a fixed assignment sequence of the variables to input terminals. He uses a matrix of

binary representations of the minterms of the truth function. His method finds all the cascades that realize a given function. An algebraic realizability test and synthesis procedure was suggested by Levy Winder and Mott (4).

In 1964 R. C. Minnick (8) presented a work which describes another method for synthesis. The logic structure which he uses is a two-dimensional array of cells. In this arrangement, each column is a "Maitra Cascade" which produces one part of the desired output. The lowest row is also a "Maitra Cascade" which achieves the desired output function by operating on the output of the columns which are its input variables (Fig. 2). He assumes a fixed ordering of the input variables and devotes part of his work to the electronic aspects of manufacturing a cell which can be adjusted to perform any desired function (Cutpoint cell).

Further work in this direction has been done by A. Mukhopadhyay (10), who developed minimization algorithms for cellular arrays.

A very recent work by C. D. Weiss (15) suggests another method of realising a function with arrays. His approach uses a combination of the "Maitra Cascade" and the "Minnick Rectangular" (Fig. 3) and he starts with the prime implicants of the function to be synthesized.

A different type of cell which has 3 inputs and 2 outputs was considered by M. Yoeli (16). This approach results in a two-rail cascade.

It is worthwhile to note that a very good survey of cellular research is presented in reference 9.

CHAPTER TWO

2.1 THEORY OF DECOMPOSITION

The method suggested in Chapter 3 of this thesis implies the use of the theory of decomposition introduced by R. I. Ashenhurst (1).

The following is an outline of the basic theorems of the theory of decomposition (taken from reference 1).

Theorem 1: A switching function $f(X_1,X_2,\ldots,X_n)$ possesses a simple decomposition in terms of $F(Y_1,Y_2,\ldots Y_s)$ and $\phi(Z_1,Z_2,\ldots,Z_{n-s})$ if and only if its $Y_1Y_2\ldots Y_s|Z_1Z_2\ldots Z_{n-s}$ partition matrix has column multiplicity $v \leq 2$.

Theorem 2: Let $f(A,B,C)$ be a function with no vacuous variables.

If $$f(A,B,C) = F[\xi(A,B),C] = G[\phi(A),B,C] ,$$

then $$f(A,B,C) = F[\eta[\phi(A),B],C] ,$$

where $\eta(\phi,B)$ is a uniquely determined function of ϕ,B, in which no variables occur vacuously, and for which

$$\eta[\phi(A),B] = \xi(A,B).$$

Theorem 3: Let $f(A,B)$ be a function with no vacuous variables.

If $$f(A,B) = F[\phi(A),B] = G[\chi(B),A] ,$$

then $$f(A,B) = H[\phi(A),\chi(B)] ,$$

where $H(\phi,\chi)$ is a uniquely determined function of ϕ,χ in which no variables occur vacuously.

Theorem 4: Let $f(A,B,C)$ be a function with no vacuous variables.

If $f(A,B,C) = F[\phi(A),B,C]$
$= G[\chi(B),A,C]$,

then $f(A,B,C) = H[\phi(A),\chi(B),C]$,

where $H(\phi,\chi,C)$ is a uniquely determined function of ϕ,χ,C, in which no variables occur vacuously.

Theorem 5: Let $f(A,B,C)$ be a function with no vacuous variables.

If $f(A,B,C) = F[\xi(A,B),C]$
$= G[\eta(A,C),B]$,

then $f(A,B,C) = H[\phi(A),\chi(B),\psi(C)]$,

where $H\ \phi,\chi,\psi = \phi*\chi*\psi$, the operation * is uniquely determined as ., +, or Δ, and the functions $\phi(A)$, $\chi(B)$, and $\psi(C)$, in which no variables occur vacuously, are uniquely determined if * is . or +, and to within complementation if * is Δ.

Theorem 6: Let $f(A,B,C,D)$ be a function with no vacuous variables.

If $f(A,B,C,D) = F[\xi(A,B),C,D]$
$= G[\eta(A,C),B,D]$

then $f(A,B,C,D) = H[\lambda[\phi(A),\chi(B),\psi(C)],D]$,

where $H(\lambda,D)$ has no vacuous variables and is determined to within complementation of the variable $\lambda, \lambda(\phi,\chi,\psi)=\phi*\chi*\psi$, and the functions $\phi(A)$, $\chi(B)$, $\psi(C)$, in which no variables occur vacuously, are determined to within complementation.

Theorem 7: Let $\delta_1,\delta_2,\ldots,\delta_p$ be the maximal decompositions of a function $f(x_1,x_2,\ldots,x_n)$ for which $n > 1$ and no variables occur vacuously. Then either (1) the secondary sets $S_1,S_2,\ldots,S_p$ are all disjoint, so that $\delta_i \leftrightarrow \delta_j$, for all $i,j \leq p$, and $S_1 \cup S_2 \cup \ldots \cup S_p = I$, or (2) the secondary sets $S_1,S_2,\ldots,S_p$ are all conjoint, so that $\delta_i \leftrightarrow \delta_j$

directly for all $i,j \leq p$, but the secondary set complements $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_p$ are all disjoint, and

$$\bar{S}_1 \cup \bar{S}_2 \cup \ldots \cup \bar{S}_p = I.$$

Theorem 8: Given a function $f(x_1, x_2, \ldots, x_n)$ in which no variables occur vacuously, the set E_f of its explicit decompositions, partially ordered by $\geq$, forms a lattice.

Theorem 9: Given a function $f(x_1, x_2, \ldots x_n)$ in which no variables occur vacuously, the set D_f of its simple decompositions, partially ordered by $\geq$, forms a lattice.

2.2 SOLUTION OF BOOLEAN EQUATIONS

The theorems presented in Section 2.1 may be expressed in the form of a set of Boolean equations. This reformulation will turn the decomposition of switching functions into the solution of simultaneous Boolean equations. There are a few methods for solving systems of Boolean equations. One of them is the map method which was developed by A. Svoboda (14). In this method, the system of Boolean equations is mapped onto a logical space, called a discriminant, which represents the validity of the system. The discriminant is a rectangular map in which the columns contain all the states of the known variables and the rows contain all the possible combinations of the unknown variables. Thus, to every entry A_{ij} in the rectangular map, which represents one combination of all the variables, a value of "1" or "0" is assigned, depending upon the validity or non-validity (respectively) of the system of Boolean equations for that particular state of variables. The discriminant gives the nature and number of existing solutions satisfying the given system.

EXAMPLE:

X1, X2 - Known Variables
Y - Unknown Variable;
All the expressions $Y = Y(X1, X2)$ satisfying the system of Boolean equation $X1 + X2\bar{Y} + \bar{X}2Y = 1$ are sought.

The discriminant of that system is:

X2X1 / X	00	01	10	11
	0	1	2	3
Y				
0	0	1	1	1
1	1	1	0	1

A_{31}

NOTE: The method used to decompose the discriminant and to find each solution is described in Section 2.4

This procedure for finding the discriminant may be done either manually or using a computer. For problems with a small number of variables, the manual method is convenient. However, when the number of variables increases, the number of entries A_{ij} in the discriminant also increases and makes the manual solution very slow and tedious. In such cases, a computer should be utilized.

2.3 SVOBODA'S BOOLEAN ANALYZER

One of the tools used in the Synthesis procedure suggested here is the Boolean Analyzer (B.A.) first proposed by A. Svoboda (14). The B.A. is a hardware unit, operating as part of a digital computer. It is capable of computing the discriminant

of a large system of Boolean equations. Its advantage is that it can process many terms of Boolean Algebra simultaneously, thus reducing the operation time.

The B.A. has two modes of operation. In the first mode, it determines the prime implicants of a given Boolean function. In the second mode, it finds the discriminant of a set of Boolean equations.

There are three fundamental theorems on which the operation of the B.A. is based. In order to understand them, some definitions must be made.

Let $y = 1$ be a given Boolean function and Y be its complement, presented as a sum of terms t_h.

$$Y = \sum t_h = 0$$

Each term t_h is of the form:

$t_h = \ddot{X}_n \ddot{X}_{n-1} \cdots \ddot{X}_2 \ddot{X}_1$ where $\ddot{X}_i$ takes one of the values 1, Xi or $\bar{X}i$ exclusively.

There are 3^n possible Boolean combinations of the form t_h, all of which make up the logical space T.

The subscript h, which is the identifier of the element t_h of the space T, takes on values that correspond to the variable X in the following way:

h_i	0	1	2
$\ddot{X}_i$	1	X_i	$\bar{X}_i$

$$h = h_1\, 3^0 + h_2\, 3^1 + \ldots\ldots + h_n 3^{n-1} = \sum_{i=1}^{n} h_i\, 3^{i-1}$$

Since each term t_h implies Y, it is a non-implicant of $y(y=\bar{Y})$. In order to determine all implicants of y it is sufficient to cancel from the 3^n-space T of all possible terms those which are non-implicants of y.

Theorem 1: Given a Boolean function y and its complement Y, let $[t_h]$ be the set of implicants of Y, and $[t'_h]$ represent all terms in T. The sufficient condition for a definite t'_h ($h' = \sum_{j=1}^{n} h'_j\, 3^{j-1}$, $t'_h \in [t'_h]$) to be a non-implicant of y is that for at least one t_h-term of Y ($h = \sum_{j=1}^{n} h_j\, 3^{j-1}$, $t_h \in [t_h]$) $hj + h'j \neq 3$ for $j = 1,2,3,\ldots,n$.

Theorem 2: Ordering of Implicants. If a term $ta \Rightarrow tb$, then $a \geq b$.

Example: $t_1 = X_1$ $t_7 = \bar{X}_2\, X_1$; $t_7 \Rightarrow t_1$ $7 > 1$

Theorem 3: Exclusion of non-prime-implicants. If a term t_h, $t_h \epsilon Tc$, is a prime implicant of y, then any other term tk implicant of t_h is not a prime implicant of y.

Example: Given $y = \bar{X}1\ X2 + X1\ \bar{X}2 + \bar{X}2\ X3$ (n=3)

We find $Y = \bar{y} = (X1 + \bar{X}2)(\bar{X}1 + X2)(X2 + \bar{X}3) =$

$X1\ X2 + \bar{X}1\ \bar{X}2\ \bar{X}3 = ta + tb$

				3^0	3^1	3^2	
	$\ddot{X}1$	$\ddot{X}2$	$\dot{X}3$	h1	h2	h3	h
ta	X1	X2		1	1	0	4
tb	$\bar{X}1$	$\bar{X}2$	$\bar{X}3$	2	2	2	26

$Y = t_4 + t_{26}$

Space T and $[t_h]$ are represented in Table I.

Table II shows the application of Theorem 1, step by step, the implicants of y ordered as per Theorem 2.

Table III shows the exclusion of non-prime-implicants. The term with the lowest identifier (t5) is always a prime implicant (because of the ordering according to Theorem 2).

To summarize, the selection of the prime implicants of y is done in the following three basic steps:

(a) Determination of the ordered set Tc containing all the implicants of y.

(b) The term tp, $tp \epsilon Tc$, with smallest identifier p is a prime-implicant of y and is transferred to the set $[t_h]$. (This set is empty at the start of the solution and contains all the prime implicants of y at the end.)

(c) All implicants of tp are cancelled in Tc.

Steps 2 and 3 are repeated until Tc is empty.

2.4 DECOMPOSITION OF THE DISCRIMINANT

The discriminant of a system contains all the solutions of a system. The number of solutions is obtained as follows: Let Ux represent the count of all non-zero elements in a certain column X of the discriminant. The total number of

solutions S is the product of all Ux integers.

$$S = \prod_{X=0}^{2^n - 1} Ux$$

If $S \neq 0$, every solution may be obtained by decomposing the discriminant into S maps Ds so that every $Ds \Rightarrow D$ but contains only one non-zero element in each column.

To explain the rules of the discriminant decomposition, note the following:

(a) Each Boolean function in the form $Yj = Yj(X1,X2,\ldots,Xn)$ $j = 1,2,\ldots,m$ (where Xi is the known variable and Yj is the unknown variable) must have a unique value for every given combination of known variables (Xi).

(b) Only a single point corresponds to that combination in the column X (known variables).
Each map Ds is thus a truth table of one solution, from which an algebraic expression can be obtained.

EXAMPLE 1

Given a system $\bar{X}1\ \bar{Y}2 + \bar{X}2X1Y1 + \bar{X}2\ Y2\ Y1 + X2\ Y2\ \bar{Y}1 = 0$

X1;X2 - Known Variables
Y1;Y2 - Unknown Variables

Its discriminant is:

Y2Y1=Y \ X1X2 / X	00 0	01 1	10 2	11 3
00 0	0	0	0	1
01 1	0	1	0	1
10 2	1	0	0	0
11 3	0	0	1	1

Ux =	1	1	1	3

S = 1 x 1 x 1 x 3 = 3

There are three possible solutions.

The 3 Decomposition maps and solutions are:

1.

Y \ X	0	1	2	3
0	0	0	0	1
1	0	1	0	0
2	1	0	0	0
3	0	0	1	0

D_1

Truth Tables.

X2 \ X1	0	1
0	0	1
1	1	0

$Y1 = \bar{X}2X1 + X2\bar{X}1$

X2 \ X1	0	1
0	1	0
1	1	0

$Y2 = \bar{X}1$

2.

Y \ X	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	1	0	0	0
3	0	0	1	0

D_2

Truth Tables.

X2 \ X1	0	1
0	0	1
1	1	1

$Y1 = X1 + X2$

X2 \ X1	0	1
0	1	0
1	1	0

$Y2 = \bar{X}1$

3.

Y \ X	0	1	2	3
0	0	0	0	0
1	0	1	0	0
2	1	0	0	0
3	0	0	1	1

D_3

Truth Tables.

X2 \ X1	0	1
0	0	1
1	1	1

$Y1 = X1 + X2$

X2 \ X1	0	1
0	1	0
1	1	1

$Y2 = \bar{X}1 + X2$

This method of decomposition may, in some cases, yield a very large number of solutions. Since, in these cases, it is a long procedure to go through every solution of the system, a method first proposed by M. Marin (6) was developed and utilised here. The basic idea behind this method is that usually only a few solutions out of all the possible ones are needed. There are, in most practical cases,

some factors which are not represented in the system of equations. These special restrictions can be presented in a form of an additional equation. Since both the original system and the constraint have to be fulfilled, the discriminant of each of them can be found (note that the column and row identifiers of both discriminants have to be identical) and both discriminants can be logically "ANDed". This is done by performing a logical "AND" on each entry Aij of both discriminants. The resulting discriminant contains the solution (s) (if any) which satisfies both the original system and the contraint.

EXAMPLE 2

To the system of Example 1, the following constraint is added: $X1\ X2\ \bar{Y}2 = 0$

The discriminant of the constraint is:

Y2Y1 \ X2X1	00	01	10	11
00	1	1	1	0
01	1	1	1	0
10	1	1	1	1
11	1	1	1	1

Ux = 4 4 4 2 ; S=4x4x4x2=128

The two discriminants are "ANDed" and the following discriminant results:

Y \ X	0	1	2	3
0	0	0	0	0
1	0	1	0	0
2	1	0	0	0
3	0	0	1	1

U_X = 1 1 1 1 ; S=1x1x1x1=1

There exists only one solution which satisfies both equations:

$$\begin{cases} \bar{X}1\ \bar{Y}2 + \bar{X}2\ X1\ \bar{Y}1 + \bar{X}2\ Y2\ Y1 + X2\ Y2\ \bar{Y}1 = 0 & \text{(3 solutions)} \\ X1\ X2\ \bar{Y}2 = 0 \quad \text{(Constraint)} & \text{(128 solutions)} \end{cases}$$

The solution is: $\begin{cases} Y1 = X1 + X2 \\ Y2 = \bar{X}1 + X2 \end{cases}$ (Solution No. 3 in previous example)

This method proves to be very useful not only in order to limit the number of solutions in a case which has a large number of solutions, but also when a system of equations has to be solved using the B.A. Since the B.A. can find the discriminant of only one equation at a time, the discriminant of each equation in the system can be found, and then all the resulting discriminants can be logically "ANDed". This is shown in Example No. 1 of Chapter 4. This method eliminates the preliminary manual preparation of transforming the system of equations into a single equation, as was done in reference 6, page 11.

2.5 BOOLEAN ANALYZER SIMULATOR

The synthesis procedure proposed in this thesis implies the use of a B.A. Since, at the present time, this is not available, a Boolean Analyzer Simulator (B.A.S.) was used instead. The first version of this simulator was programmed by M. Marin (6) for the SIGMA 7 computer. This program was written in FORTRAN IV except for 3 subroutines which were written in assembler language. The program was limited to the size of the UCLA SIGMA 7 computer (32K words of memory) thus having a maximum of twenty-two variables and five hundred terms. Since it was felt that the B.A.S. should be independent of any particular machine, it was modified in the course of this work. The entire program is now written in FORTRAN IV, and changes were made so that the dimensions can be very easily adapted to any size acceptable to the machine used. FORTRAN IV was chosen because this is the most universal scientific computer language in North America today, and almost all computer installations have a FORTRAN IV compiler.

Since the original B.A.S. was modified, changes were also made to widen and extend its scope. The following are the basic changes which were implemented:

1. The entire program is written in FORTRAN IV, and is thus "machine-independent". The major assumptions made were that a fixed point number is represented in a 32-bits word, and that a negative fixed point number is stored in a "two's complement" form in the computer memory. These assumptions will suit most of the available commercial machines.
2. Arrays were classified into blocks within adequate common statements for flexibility in calling subprograms. The upper limits of iterations in DO loops were specified by an integer-valued variable, which is common to all subprograms.

3. The format for printing the discriminant was changed to have the form which is described in Section 2.2.
4. An option was added which will enable working with equations of the form G = 1 as well as G = 0. This is important especially when the prime implicants of a function are sought, because the necessity of inverting the function prior to its processing by the computer is eliminated.
5. B.A.S. was turned into a subroutine which is called whenever the B.A. should have been used. This enables the utilization of B.A.S. in future applications.
6. An additional subroutine DECOMP was written, which decomposes a discriminant and finds all its solutions. Its theoretical basis is described in Section 2.4 and its corresponding flow chart is shown in Fig. 4.

This subroutine (DECOMP) makes extensive use of the identifiers of the rows and columns of the discriminant because their binary representation displays their corresponding minterm in the truth table. These identifiers are used as array subscripts; variable names, CLM and RAW, stand for the columns and the rows respectively. Since zero is not a legitimate subscript, one (1) was added to each identifier and then subtracted whenever a translation to binary was made. Two arrays of dimensions 2^N (N = number of known variables), namely, D and P, are used. D acts as a solution counter while P stores the decomposed discriminants one at a time (each one represents one complete solution). D (CLM) stores the number of times a "1" appears in column CLM, namely Ux. This is done for all the columns of the discriminant.

Once D (CLM) is obtained for all columns, all its terms are multiplied to yield and display the number of solutions S.

$$S = \prod_{CLM=1}^{2^N} D(CLM) \qquad (D(CLM)=Ux)$$

If S = 0 or S > MAX, where MAX stands for the maximum number of solutions imposed by the user (see Section 3.1), the process is stopped. Otherwise, the actual decomposition starts.

The discriminant of a particular solution is stored in array P in the following way: in each column of the discriminant there is only one element which equals 1 (all the other elements in the same column equal 0). The row identifier of that element is stored in P (CLM) (CLM=1,2,...,2^N).

In order to express each one of the unknown variables as a function of some known variables, more processing (following Svoboda's Algorithm) is necessary. The row identifier which is included in each element of P is translated to its minterm representation and is checked. If it includes the particular unknown variable (which is sought at that time, since each variable is processed separately) in an uncomplemented form, then the corresponding column is an implicant of the unknown variable. Once all the columns are tested, all the minterm implicants of the unknown variable are obtained. Using this result, B.A. S. finds the prime implicants of that unknown, and the result is printed.

In order to go through all the possible solutions, a methodical sequence of decomposition is required. The first solution will be composed of the highest located "1" on each column. For each of the following solutions,

P(1) will be changed to the next '1' downwards in the same column until no more '1's are available. After having found all the solutions in which the '1's of the first column only were permuted, the same is done for the second column, then the third column etc. until the 2^Nth column is reached. For each '1' processed from the ith column (i=2,...,2^N) all the permutations of the '1's in the 1st, 2nd,...,(i-1)th columns have to be repeated, so that if for example D(1)=3, each '1' in the second column (CLM=2) will yield 3 solutions.

Two column indices, I and CLM, are used to keep the sequence. The index CLM identifies the column which is processed for the first time and I identifies the column to the left of CLM whose '1's are being permuted. After the lowest '1' in a column has been used, the program returns to the highest located '1' in that column and starts permuting the next column to the right (I=I+1).

After finishing the permutations of column I which is next to CLM (I=CLM-1), the following actions take place:

1. D (CLM) = D (CLM)-1
2. P (CLM) is changed to include the row identifier of the next '1' downwards in CLM.
3. I is set to 1 (I=1).

Then the procedure is repeated again. Whenever all the '1's of a column CLM have been processed, D(CLM)=0 and CLM is stepped up one unit to the next column (CLM=CLM+1).

This procedure is iterated until the whole array D equals zero; i.e. D(CLM)=0 for CLM=1,2,...,2^N. Example 1 demonstrates this procedure.

EXAMPLE 1: Given the same system as described in Example 1 of Section 2.4, its discriminant was found to be:

Y2Y1 \ X2X1	00	01	10	11	RAW
0=00	0	0	0	1	1
1=01	0	1	0	1	2
2=10	1	0	0	0	3
3=11	0	0	1	1	4
CLM	1	2	3	4	
D =	1	1	1	3	; S=1x1x1x3=3

SOLUTION NO. 1 (Which corresponds to D1 in Section 2.4)

P	3	2	4	1
CLM	1	2	3	4

Y2=1 in Rows 3 and 4 therefore:

CLM	1	2	3	4
X2X1	00	01	10	11
Y2	1	0	1	0

$Y2=\bar{X}2\bar{X}1 + X2\bar{X}1 = \bar{X}1$

Y1 = 1 in Rows 2 and 4 therefore:

CLM	1	2	3	4
X2X1	00	01	10	11
Y1	0	1	1	0

$Y1 = \bar{X}2X1 + X2\bar{X}1$

After printing the first solution, array D changes in the following way:

D	0	1	1	3

D	0	0	1	3

D	0	0	0	3

While P stays as was.

Then array P changes and the second solution is obtained.

SOLUTION NO. 2 (Which corresponds to D2 in Section 2.4)

P	3	2	4	2

CLM	1	2	3	4
X2X1	00	01	10	11

Y2	1	0	1	0

$Y2 = \bar{X}2\bar{X}1 + X2\bar{X}1 = \bar{X}1$

CLM	1	2	3	4
X2X1	00	01	10	11

Y1	0	1	1	1

$Y1 = \bar{X}2X1 + X2\bar{X}1 + X2X1 = X2 + X1$

After printing the second solution, array D changes:

D	0	0	0	2

Then array P changes to yield the third solution.

SOLUTION NO. 3 (Which corresponds to D3 in Section 2.4)

P	3	2	4	4

CLM	1	2	3	4
X2X1	00	01	10	11

Y2	1	0	1	1

$Y2 = \bar{X}2\bar{X}1 + X2\bar{X}1 + X2X1 = \bar{X}1 + X2$

CLM	1	2	3	4
X2X1	00	01	10	11

Y1	0	1	1	1

$Y1 = \bar{X}2X1 + X2\bar{X}1 + X2X1 = X2 + X1$

After printing the third solution, the following happens:

D =	0	0	0	1
D =	0	0	0	0

and the decomposition is terminated.

CHAPTER THREE

3.1 THEORY AND ALGORITHMS

In the synthesis procedure outlined below, the cells utilized have the following characteristics (Fig. 5):

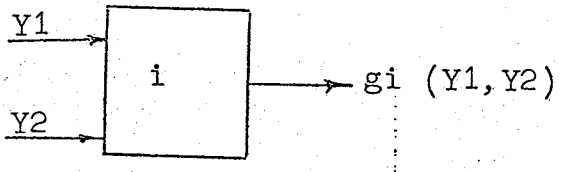


Fig. 5

1. Each individual cell has 2 inputs and 1 output.
2. Each individual cell can perform one Boolean function within a complete set of functions.

The complete set of 5 functions (reference 15, p. 187, reference 7, p. 7) which was assumed is:

Index Number i	Function gi
1	$Y1 + Y2$
2	$\bar{Y}1 + Y2$
3	$\bar{Y}1\ Y2$
4	$Y1\ Y2$
5	$Y1\ Y2 + \bar{Y}1\ \bar{Y}2 = \bar{Y}1 \oplus Y2$

Note that any other complete set of functions could have been used without affecting the method; therefore, it is claimed that the synthesis methodology presented here is completely general.

Let X1, X2, ...,Xn be the input variables to the network - the "Signal Inputs".

Let F (X1, X2,...., Xn) be the desired output function.

Let the "Last Cell" of the network be the cell whose output coincides with the sought Function F.

Let Y1, Y2 be the inputs to the "Last Cell" of the network.

Let gi be the function performed by the "Last Cell" of the network.

The logical relation representing the conditions for the "Last Cell" is:

F (X1, X2,....,Xn) = gi (Y1, Y2) (i = 1, 2, 3, 4, 5) (1)

in which Y1, Y2, are the unknowns.

Let the problem be to synthesize F with ℓ Logic Levels ($\ell \leq$ maximum number of logic levels of the net).

The procedure to be followed is:
Equation (1) is solved for i= 1, 2, 3, 4, 5, thus finding the inputs Y1, Y2 to the cell for each possible function gi that it can perform. Once all the solutions are achieved, any one can be chosen, determining the cell function gi which gives the expected results.

Each one of the two input functions Y1 (X1, X2,...,Xn) and Y2 (X1, X2,....,Xn) thus obtained may be either a logical constant or a "signal input" or a function Y. If Y is a function realizable by (ℓ -1) logic levels, then the cell whose output coincides with Y may be sought in exactly the same manner, namely, by setting Y to be the output function of this cell. If, on the other hand, Y is found not to be realizable by (ℓ -1) cellular logic levels, then another solution is selected and tested. The entire procedure is iterated for functions Y1 and Y2 until the Lth logic level is reached. If, at this stage, no acceptable solution is found, it is concluded that no realization exists with ℓ-logic levels. The flow

chart describing the algorithmic steps of this synthesis methodology is shown in Fig. 6.

The criterion according to which the cell function (gi) is selected may vary according to the particular case. If desired, all the possible solutions can be displayed, so that the selection may be made manually by the user. This may be ideal when an "on-line" interaction with the B.A. system is available.

The basic part of the synthesis method outlined above (see Section 4.1) was translated into a computer program written in FORTRAN IV. The program solves the equation $F = gi$ and finds gi, Y1, Y2 for each $i(i = 1, 2, 3, 4, 5)$. No further selection is carried out in the general program because of the reasons described in Section 1.1. (Selecting a "two-level logic" by the use of the constraints is demonstrated in Section 4.2).

The computation steps of this program are:

1. For each $gi(i = 1, 2, 3, 4, 5)$, the equation $F = gi$ is turned into the form $\bar{F}\, gi + \bar{F}\, gi = 0$ which is acceptable to either the B.A. or B.A.S. (From here on, B.A.(S.) will stand for the use of either the B.A. or the B.A.S.) This is done using five (5) separate subroutines, one for each cell function (gi), in order to enable easy changes in the set of functions used. The complement of F ($\bar{F}$) is achieved by the B.A.(S.) operating in mode 1.
2. The discriminant of the system $F = gi$ is computed by the B.S.(S.) operating in mode 2 (for $i=1, 2, \ldots, 5$).
3. The discriminant is decomposed by "DECOMP". The number of solutions is printed, and each one is computed and displayed. This is one point in which "on-line" interaction is desirable, to determine how to proceed

once the number of solutions is known. Since such a feature is not available for this work, the program does not find the solutions if their number exceeds a prescribed value (MAX). The constraints which are described in Section 2.4 may be used in steps 2 and 3. If a constraint exists, its discriminant is determined by the B.A.(S.) and logically "ANDed" to the one of the system. The new discriminant is decomposed by "DECOMP" which prints the number of the solutions and each solution in full.

3.2 PROCEDURE

The flow chart of the computer program is shown in Fig. 7. The synthesis procedure as described in Section 3.1 is carried out in loop B, and the computation blocks preceding this loop are used to make the necessary initial preparations.

Thus, in block I the terms of the function F are read and stored for future processing; in block II the B.A.(S.) (operating in mode 1) is used to find the complement of the function F ($\bar{F}$). The prime implicants of $\bar{F}$ are then stored in the same fashion as the terms of F. If there are constraints to the system, their equations are read in block III, and their terms are stored in the same way as is done in block I. Each constraint is referred to by an address MJ which indicates the order of appearance.

After storing the terms of $\bar{F}$, F, and the constraints (where applicable), the program searches for possible solutions with the first selected cell. The program then will try to realize the output function F, using this cell.

If constraints exist, it will try to realise F, satisfying these constraints, one by one, using only the selected cell. This is done in loop A (fig. 7). When all the solutions using one particular cell have been found, the next cell is selected and the same procedure is repeated. This is done (loop B) until the complete set of cells (five cells) is tested.

Block IV consists of five (5) different subroutines, each one corresponding to one cell type. The program is channelled to the appropriate subroutine by the choice of the cell (i). Each subroutine uses its particular cell function gi(i = 1, 2, 3, 4, 5) to convert the equation F = gi into the form $\bar{F}gi + \bar{F}gi = 0$ which is acceptable to the B.A.(S.).

Block V processes the terms of the function $\bar{F}gi + \bar{F}gi = 0$ using the B.A.(S.) (operating in mode 2). The B.A.(S.) finds the discriminant of that equation, which represents the existing (if any) realization of F with cell i. If the synthesis of F is subject to constraints, then the set of simultaneous equations:

$$\left\{\begin{array}{l} F = gi \\ \text{constraint equation MJ} \end{array}\right\} \text{will be solved.}$$

In subroutine CSTRT (Block VI), each constraint is added independently to equation F = gi. In this subroutine, the discriminant of a constraint MJ is obtained with the aid of the B.A.(S.). The two discriminants, one expressing the desired function realized by cell i and the other expressing the MJ's restriction, are logically "ANDed". The resulting discriminant is then processed and decomposed by subroutine DECOMP (Block VII Fig. 7). If no constraints exist, the original discriminant obtained in Block V is sent directly to DECOMP, bypassing subroutine CSTRT.

Subroutine DECOMP (Block VII) finds S = the number of existing solutions in the particular case. If S > MAX, no solution is sought since MAX predefines the upper limit of solutions to be retrieved. Otherwise, the discriminant is decomposed and all the solutions obtained are printed.

After completing the process with cell i and constraint MJ, the procedure is repeated with the same cell and the next constraint until all the constraints have been tried out. This is described in loop A. Once all the constraints have been processed, MJ is set back to 1 and the next cell is selected (by stepping i). The actions of loop A start again with the new cell (loop B) and continue until no more cells are available ($i > 5$). When $i > 5$, the synthesis of that problem has been terminated and the next problem can be read in and solved.

User Instructions are presented in Appendix E.

CHAPTER FOUR

4.1 PROGRAM EVALUATION AND LIMITATIONS

The computer program developed in this thesis and described in Section 3.2 is listed in Appendix A, and was tried with a few examples (see Section 4.2). These trials proved the program to be capable of solving the suggested problem (see Section 3.1), i.e., for a given Boolean output function, the program selects, out of a set of logical 2-input and 1-output cells, those which are able to yield that function at their output. It is also capable of finding the two-cell inputs which will yield the desired function at the cells output. In the following discussion, one such cell and its two inputs (Y1, Y2) will be referred to as one solution.

The computer program can find all the existing solutions for a particular function with the given complete set of cells, including the most complicated and impractical ones (see Section 4.2).

In order to select only the adequate solutions, some constraints were introduced which the program proved to handle correctly. With a slight modification, the program was made to select solutions which comply with two different restrictions simultaneously (see Example 3, Section 4.2).

The disadvantage of the method for selecting a solution with constraints is that the constraints have to be expressed in the form of Boolean functions. The expression of the selection criterion in the form of equations (constraints) may result in a very large number of Boolean equations, thus making the computer operation long and costly.

Another possible method for selecting the solutions is to find all the existing solutions, and then check each one

against a certain criterion. This is practical only when the number of solutions is relatively small and the processing time is not too long. (The number of solutions which is feasible to test depends on the processing time.) The reason for using the constraint method for solutions selection in the general program is that it does not depend on the total number of solutions.

Subroutine "COUNT" is used because the McGill University IBM 360/75 Computer System, which was used for these trials, has no automatic page-skipping feature. Another limitation was imposed on the program because no explicit overflow protection message was available in the installation used.* For this reason, whenever the number of solutions exceeds 10,000, a message is printed and no solution is retrieved (see Example 2, Section 4.2). The program's adaptability to different sets of cells was demonstrated when a change in the set of cells used was tried out. The cell function g_5 was changed in Example 3, Section 4.2, to $\bar{Y}1Y2 + Y1\bar{Y}2$ from $\bar{Y}1\bar{Y}2 + Y1Y2$ which was used in Examples 1 and 2, Section 4.2. Thus, the same program can be used employing different cells or for the purpose of selecting an optimal set of cells for any particular use.

A limitation of the program is the length of its processing time. Since, when working with a computer, the cost is determined mainly by the processing time, the program is considered to be too expensive for many possible uses. Because of this fact, no computer program was written for the complete synthesis method which was suggested in

NOTE: * When an overflow occurs, a large negative number is retrieved.

in this thesis (Section 3.1, Fig. 6). It is assumed, however, that if a B.A. had been used, as was originally suggested, this problem could have been eliminated. To explain this assumption, consider the comparison in Reference 6, page 125, between the processing times of the B.A. and the B.A.S. It was found there that the time ratio between the two is of the order of 10^4. If a function with only one constraint (expressed in the form G = 1) is to be realized using the program presented here, the B.A.(S.) is called twelve (12) times plus two times for each solution. Taking into account the processing time ratio between the B.A. and the B.A.S., it is evident that the use of the B.A.S. makes the program very expensive.

It is strongly recommended that subroutine DECOMP be turned into a hardware unit as well. This would be a natural extension of the B.A., since a discriminant is not usually considered as a final answer and its decomposition always follows. Subroutine DECOMP processes the discriminant which was found by the B.A.(S.) and uses the B.A.(S.) to find the prime implicants of each solution; therefore, it would be a natural part of the B.A. itself.

Assuming that the time ratio between a software and a hardware decomposition unit is of the same magnitude as the time ratio between the B.A.S. and the B.A., it is evident that the cost of using the synthesis program is going to drop significantly with the use of the hardware units.

4.2 PROGRAMS AND EXAMPLES

The listing of the general computer program (described in Section 3.2) which is used in Examples 1 and 2 is presented in Appendix A. The outputs for examples 1 and 2 are presented in Appendices B and C respectively. The program listing and outputs for Example 3 are presented in Appendix D. Instructions for the users are presented in Appendix E.

The results achieved are shown by using the identifier h in the following way:

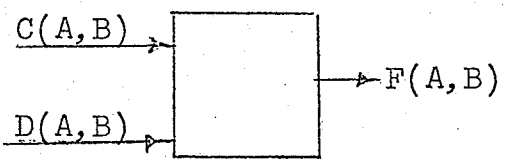
h	0	1	2
$\overset{..}{X}i$	1	Xi	$\bar{X}i$

Note that the name of each variable, that is, A, B, C, etc., is printed underneath the identifier. For example, $\overset{1201}{ABCD}$ stands for $A\bar{B}D$. MAX, which is the maximum number of solutions that DECOMP will find for any discriminant, was set to equal 4.

EXAMPLE 1

Let the function to be realized be:

$F = A\bar{B}$



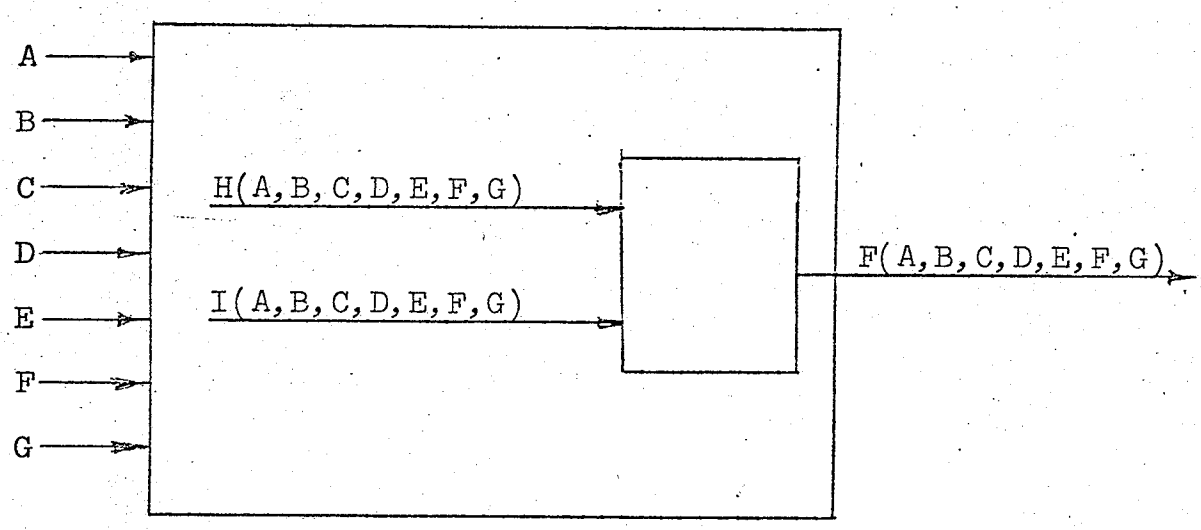
The solutions sought are either without any constraint (constraint No.1) or with the constraint C = A expressed in the form $AC + \overline{AC} = 1$ (constraint No.2). Note, for example, the solutions with cell type 4 (F = CD) and constaint No. 2 (C = A) (page B2). Solution No.2, namely,

$C = A$, $D = \bar{B}$, contains the least number of literals, and requires only the inversion of the input variable. However, we also diplay the valid solutions $C = A$, $D = A\bar{B}$; $C = A$, $D = \bar{A}B + A\bar{B}$; $C = A$, $D = \bar{A} + \bar{B}$; which obviously contain a larger number of literals and require logical operations on the input variables. A duplication, due to the symmetry of the cells used, can be seen in the solution with cell type 1 ($F = C + D$) without any constraint (page B1). In this case, solution No. 2 with only the two inputs to the cell (C and D) interchanged.

EXAMPLE 2

Let the function to be realized be:

$F = ABC + \bar{A}EG + D\bar{E} + F\bar{G}$



The solutions sought are either without any constraint (constraint No. 1) or with the constraint

$$\begin{cases} I = \bar{A}EG + D\bar{E} + \bar{A}DG \\ HI = 0 \end{cases}$$

This constraint is expressed in the form,

$$A\bar{D}I + \overline{D\bar{E}}I + \overline{\bar{D}G}I + AEI + E\bar{G}I + \bar{A}EG\bar{I} + D\overline{E\bar{I}} + \bar{A}DG\bar{I} + HI = 0$$

(constraint No.2).

Note that, when the number of existing solutions exceeds 10,000, a message to that effect is printed, and the actual number of solutions is not retrieved.

It is interesting to note that, with the increasing number of variables, the number of solutions seems to tend to either one of the two extremes: either a very large number of solutions or none at all. This means that the problem as formulated is either not sufficiently constrained or over-restricted, respectively.

The Computer's Central Processing Unit (CPU) took 3 minutes, 11.81 seconds, for compiling the program and solving examples 1 and 2. The compilation alone takes approximately 0 minutes, 16 seconds, which means that solving Examples 1 and 2 took approximately 2 minutes, 56 seconds.

EXAMPLE 3

The following example realizes functions of four (4) variables (A,B,C,D) with cellular networks which have up to two levels of logic. The program finds the cell with the two inputs (E,F) which will yield the desired function F(A,B,C,D) at its output. In order to limit the realization to consist of one or two logic levels only, each one of the two inputs to the cell (E,F) is restricted to be a constant, a signal input (A,B,C or D) or the.

output of a single logic level. The way in which this is done is the following: 46 constraints, which represent all the possibilities for E (A,B,C,D) to be either a signal input or the output of a single logic level network, are imposed on the system one by one. With every such constraint, the number of solutions S is computed. This number will include all the possible realizations, including the ones in which F does not comply with the requirements. In order to restrict the input F as well, 51 additional constraints are needed. Depending on the number of solutions computed, the program selects one of the following modes of operation.

Mode 1: If $S > 40$, the 51 restrictions which limit F will be imposed on the system one by one. Once this is done, both E and F are restricted properly and the possible solutions will be found.

Mode 2: If $S < 40$, it is more economical to find all the solutions and test F in each one. Since E is already properly restricted, the solution is rejected if F does not yield a "two-level" realization.

The program had to be changed to suit this procedure. The basic changes made are:

1. The number of variables throughout the program was fixed at four, and the dimensions were reduced respectively.
2. The constraints were read at the beginning of the program since they were common to all the functions. They were split to those that limited E and the ones that limited F.
3. The program was changed to be able to impose two different constraints on the system simultaneously (for use in Mode 1).

4. A section testing the F input was added (for use in Mode 2).
5. Cell type 5 was changed to execute an Exclusive Or. This was done to show the versatility of the program.

Since the intention in doing this program was to create a table of solutions for all the functions of 4 variables (taken from reference 3, pp. 396-407), some changes were also incorporated in order to make the running of the program more economical. The table would consist of 238 functions (and possibly their complements as well); therefore, the processing time of each function becomes very important. The time for solving examples 1 and 2 gives sufficient reason to try to reduce it. Since this program processes functions of four variables only, the memory size actually used is relatively small; therefore, some changes which saved processing time but used more computer storage were made. The changes made for this purpose are:

1. The constraints are stored in the form of their discriminants rather than prime implicants. This means using more memory for each constraint (there are 97 constraints), but saving the time of computing the discriminant whenever a constraint is used.
2. The discriminants are stored not in bit-strings but in digit-strings. For N = 4, the resulting difference in storage space is not significant. This saves the time of transforming the discriminant from digits to bits and back whenever it is needed. (Subroutines DISCRI and PUT are eliminated).
3. The program was changed to look only for the first valid solution with each function processed.

Change No. 3 was incorporated because of the results of the following experiments which were done with one arbitrary function: $F = \bar{A}B + B\bar{C}$.

When all the solutions were computed, the processing time in the computer was 36.44 seconds. For the same function, when only one solution (the first to be found) was computed, the processing time was 22.08 seconds (see both outputs in pages D30 - D35). Considering that the 22.08 seconds were spent on compiling the program, processing and storing the discriminants of the 97 constraints, and finding one solution, it is clear that to spend an additional 14.36 seconds, just to find the rest of the acceptable solutions for one function, is very costly, especially when hundreds of functions are processed. Therefore, it was decided to stop the processing of any particular function once one acceptable solution was found, until the B.A. hardware unit becomes available. The printed solutions, therefore, may not be the best ones. However, after one has prepared the complete table, one may select some of the functions which have a solution, and run the program again, this time finding all the possible solutions. The complete table is not included in this thesis; instead, a table of only 23 functions is presented in Appendix D. The processing time for compiling and computing the 23 realizations was 4 minutes, 38.32 seconds. Compared with the time required for running Examples 1 and 2, it is seen that a significant reduction in computer processing time has been achieved.

4.3 RECOMMENDATION FOR FUTURE STUDY

The studies which were done in the course of this thesis, combined with the results which were achieved, indicate that further investigations in the following directions are desirable.

1. The hardware unit for the Boolean Analyzer should be modified to include in it the decomposition of the discriminant as well. Once this has been finished and the system is built, the times for realizing systems using the procedure outlined in this thesis should be tested and compared to the results presented here. We also recommend the inclusion of an "on-line" interaction facility in the system and modification of the computer program so that it will wait for further instructions once the number of solutions is displayed.
2. When the hardware units are available, and provided that the processing time is reduced sufficiently by using them, a program along the lines described in Fig. 6, Section 3.1, for the complete general synthesis method should be written. This general program will be a most useful synthesis tool for the design of digital networks, and should prove to be helpful for realizing small systems as well as large ones.
3. Further studies on the selection of solutions should be made. The method of constraints can be improved and extended. This method will always be useful as a means of solving a system of a few equations simultaneously without any manual calculations prior to the computer processing. The other selection method of finding all the solutions and testing them one by one sould also be studied. If the hardware unit built for DECOMP

will find a number of solutions simultaneously (by parallel processing, as was suggested for the Boolean Analyzer), this method, when developed, might prove to be easier to implement and more useful than the constraints method.

4. Optimization criteria for selecting the solutions can be investigated, and several of them can be attached to the program with proper selecting facilities. These may be learned by the system as Examples are processed.

C O N C L U S I O N S

In this thesis, a synthesis procedure for cellular networks, based on a new general approach, has been proposed. The cells used are two-input one-output cells which can perform one logical function out of a given set, as described in Chapters 1 and 3.

The proposed procedure (Chapter 3) uses Svoboda's Boolean Analyzer (B.A.), but since no such unit is available at the present time, an extended version of the Boolean Analyzer Simulator (B.A.S.) was developed and utilized for the experimentation. The basic part of the procedure was programmed in FORTRAN IV and run on the McGill IBM 360/75 Computer.

The results of the experiments are reported in Chapter 4. These proved that the new general approach to this synthesis problem yields interesting and useful results. The program implementing the basic synthesis procedure finds all the existing solutions, and a method for selecting the desired solutions is made available by adequate insertion of constraint equations. The constraint equations are selected by the designer according to the particular requirements and are part of the input data. Thus, the program proper is not affected by these constraint equations.

Because of the flexibility of the procedure, it may be applied to various logic problems. For example, in Chapter 4, a table containing the two-level realizations of four-variable functions was compiled. Typically, the execution time required per function was 1.5 minutes, since the B.A.S. instead of the B.A. hardware unit was used (see Chapter 4).

The most important feature of the developed program is that it processes functions with a very large number of variables, this number being limited only by the memory of the computer used.

Program documentation, a manual for users, and program listing are presented in Appendices E and A. Experimental results are listed in Appendices B, C, and D.

REFERENCES

1. Ashenhurst, R.L., The Decomposition of Switching Functions, Proc. Intern. Symp. Theory of Switching, Harvard University, pp. 74-116, 1957.
2. Curtis, H.A., A New Approach to the Design of Switching Circuits, D. Van Nostrand Company, Incorporated, Princeton, N.J., 1962.
3. Harrison, M.A., Introduction to Switching and Automata Theory, McGraw Hill Company Incorporated, New York, 1965.
4. Levy, S.Y., Winder, R.O. and Mott, T.H. Jr., A Note On Tributary Switching Networks, IEEE Trans. EC-13, 21, pp. 148-151.
5. Maitra, K.K., Cascaded Switching Networks Of Two-Input Flexible Cells, IRE Trans. EC-11, 2, pp. 136-143.
6. Marin, M.A., Investigation Of the Field Of Problems For the Boolean Analyzer, PhD Dissertation, University of California at Los Angeles, Report No. 68-28.
7. Marin, M.A., Some New Applications Of the Boolean Analyzer, Proceedings, International Symposium on "Desing and Application of Logical Systems" September 1969, Brussels.
8. Minnick, R.C., Cutpoint Cellular Logic, IEEE Trans. EC-13, pp. 685-698.
9. Minnick, R.C., A Survey Of Micro-Cellular Research, J.A.C.M., Vol. 14, pp. 203-261.
10. Mukhopadhyay, A. Unate Cellular Logic, IEEE Trans. C-18, 2, pp. 114-121.
11. Ninomiya, I, On the Number Of Genera Of Boolean Functions Of N Variables, Mem. Fac. Eng. Nagoya Univ. Vol. 11, No. 1, pp. 54-58.

12. Skalansky, J., General Synthesis Of Tributary Switching Networks, IEEE Trans. EC-12, pp. 464-469.

13. Skalansky, J, Korenjak, A.J., and Stone, H.S., Canonical Tributary Networks, IEEE Trans. EC-14, pp. 961-963.

14. Svoboda, A., Boolean Analyzer, Proceedings of IFIPS 1968 Congress, IFIP, 1968.

15. Weiss, D.C., Optimal Synthesis Of Arbitrary Switching Functions With Regular Array Of 2-input, 1-output Switching Elements, IEEE Conference 1968, Symposium on Switching and Automata Theory, pp. 187-212.

16. Yoeli, M., A Group-Theoretical Approach to Two-Rail Cascades, IEEE Trans. EC-14, 6, pp. 815-822.