Matrix methods in combinational logic design.

Edwards, C. R.

MATRIX METHODS IN COMBINATIONAL LOGIC DESIGN.

submitted by C.R. Edwards
for the degree of Ph.D.
of the University of Bath.
1973

SUMMARY.

The object of this thesis is to present certain matrix techniques which may be employed in the analysis and synthesis of binary combinational logic circuits. These techniques are readily implemented on the digital computer.

In developing these methods care has been taken to avoid heuristic algorithms so that each technique has a firm mathematical foundation.

The first chapter of the thesis considers a Boolean matrix approach to logic analysis and synthesis. These matrices allow the rigorous and formalised representation of logic circuits. An important property of these matrices is that they embody multiple-output circuit representation and that , together with certain matrix operations , they may be used in the synthesis of multiple output circuits on an iterative basis.

The second chapter of the thesis describes a matrix transformation technique which has properties directly applicable to logic synthesis. This technique may be employed not only in the field of conventional logic design but also in the design of circuits using threshold gates. Certain transform-domain operations are used to synthesise logic circuits directly from the transformed truth-table representation of Boolean functions. These operations may also be used in the classification of Boolean functions. They may also be employed in the synthesis of multiple-output circuits and pattern recognition.

The third section of the thesis concerns itself with other research work initiated by the topics discussed in chapters one and two .Of special interest is the description of a universal threshold logic gate and its role in logic synthesis.

CONTENTS

Chapter 2. THE APPLICATION OF THE RADEMACHER/WALSH TRANSFORM TO LOGIC DESIGN AND BOOLEAN FUNCTION CLASSIFICATION.

(i) DEFINITIONS.

n	the letter n will be used exclusively to denote the order of a Boolean equation. n is the minimum number of defining variables necessary to always unabiguously represent a Boolean function of order n .
x_i , $1 \leqslant i \leqslant n$	will be used exclusively to denote the defining variables of a Boolean function of order n .
$F(x_1, x_2, \ldots, x_n)$	will denote any n th order Boolean function.
$F_i(x_1, x_2, \ldots, x_n)$	will denote a particular n th order Boolean function.
ψ_j,	a point in n-space defined as follows : Let $\langle S \rangle$ be the set of all possible unique values of the vector $x_1, x_2, \ldots, x_n]$ in the range $0, 1$; then each member of $\langle S \rangle$: s_i , $1 \leqslant i \leqslant 2^n$, is an n-tuple ψ . A particular n-tuple ψ_j , called the j th n-tuple, is defined as ψ_j , $j = 2^{n-1}.x_1 + 2^{n-2}.x_2 + \ldots + 2^0.x_n$.
True minterm	an n-tuple at which a given function has the logical value 1.
False minterm	an n-tuple at which a given function has the logical value 0.
Canonical representation	a method of representing a Boolean function where the n-tuples on which such functions are defined are always written in the same positions. The function is then said to be in 'canonical form'. This term is also applied to the positioning of the spectral coefficients of a Boolean function.

Truth table	a canonical representation of a Boolean function. Each n-tuple is tabulated together with the corresponding value of the function. The n-tuples are written in order as : $\psi_0, \psi_1, \psi_2, \ . \ . \ . \ , \psi_{2^n-1}$. See Fig. 1a and reference 1.
Karnaugh map	a canonical representation of a Boolean function. The map consists of an area divided into 2^n adjacent squares. Each square represents an n-tuple and contains a minterm. Squares with common sides differ only by a Hamming distance of one. See Fig. 1b and reference 1.
$F_{\psi_j}(x_1, x_2, \ . \ . \ , x_n)$	will denote the value of an n th order Boolean function at the n-tuple ψ_j.

(ii) LIST OF SYMBOLS USED.

In the approximate order in which they appear.

$[\ ]$	Integer matrix.
$]$	Integer vector.
$\cdot$	Logical AND operator.
$+$	Logical OR operator.
$\bar{\ }$	Logical COMPLEMENTATION operator.
$\oplus$	Logical exclusive-OR operator (Non-equivalence)
$\bar{\oplus}$	Logical not-exclusive-OR operator.(Equivalence)
$[A]$	Unit or Identity Boolean matrix.
$c_{i,j}$	Element of an integer matrix $[C]$ appearing in the i th row and j th column.
c'_j	j th column vector of an integer matrix $[C]$ expressed in decimal notation.
$\langle\ \rangle$	A set.
$\subset$	Inclusion
$\cap$	Intersection
$\cup$	Union
$[\ ]^{-1}$	Inverse Boolean matrix.
$\frown$	Tie, used to indicate related column vectors in conditionally related matrices.
$[\ ]^{\pi}$	Matrix raised to exponent π.
$[\ ]^{\frac{1}{R}}$	R th root of a matrix.
$[\emptyset]$	Operator matrix.
$\$$	Parallel composition operator.
$[T]$	Rademacher/Walsh transform matrix.
$\langle R\rangle$	Set of spectral coefficients or spectrum.
$[\Lambda]$	Matrix in Galois Field 2
$\lvert\Lambda\rvert$	Determinant in Galois Field 2.
$\hat{e}$	Expected value.

CHAPTER 1.

Boolean Matrices.

1.1 Introduction.

The type of Boolean matrices described here were first developed by J.O.Campeau in the late 1950's, see references 2,3 and 4.

Campeau was particularly interested in using these matrices in the analysis and synthesis of counting circuits and for this reason considered matrices of dimension $n \times 2^n$ almost exclusively.

These matrices, whilst having properties analogous to those of conventional matrices , both in terms of structure and algebra, may be applied directly to the analysis and synthesis of logic circuits. They are particularly useful in the representation of cascaded multiple-output logic modules and have associated operations which are easily implemented on the digital computer.

1.2 Basic Concepts.

1.2.1 Representations.

Consider the representation of algebraic equations under conventional matrix algebra :

$$\begin{bmatrix} \text{Coefficient} \\ \text{Matrix} \end{bmatrix} \begin{bmatrix} \text{Defining} \\ \text{Variables} \end{bmatrix} = \begin{bmatrix} \text{Required} \\ \text{Functions} \end{bmatrix} \quad . \quad . \quad (1.1)$$

It will be recalled that the coefficients are arranged in a particular order so that,under matrix multiplication,the correct coefficient is associated with a particular variable , e.g. :

$$\begin{bmatrix} 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} P \end{bmatrix}$$ defines a single function P where

$P = 3x_1 + 2x_2$. Similarly $\begin{bmatrix} 3 & 2 \\ -4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} P \\ Q \end{bmatrix}$ defines

two equations P,Q where $3x_1 + 2x_2 = P$

and $-4x_1 + x_2 = Q$.

Now there is no reason why Boolean equations should not be

FIG. 1a: TRUTH TABLE

φ_j	x_1	x_2	x_3	x_4	$F(x_1,x_2,x_3,x_4)$
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

FIG. 1b.: KARNAUGH MAP

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	1
10	0	0	1	1

x_4

x_3

x_1

x_2

$F(x_1,x_2,x_3,x_4)$

represented in a similar way.

Consider $\begin{bmatrix} c_1 & c_2 & \cdot & \cdot & c_i & \cdot & \cdot & c_{2^n} \end{bmatrix} \left.\begin{matrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{matrix}\right] = U \Big] .$.(1.2)

where $x_1, x_2, \ldots, x_n$ are the defining variables of a Boolean function $U = F(x_1, x_2, \ldots, x_n)$ and the coefficients $c_1, c_2, \ldots c_{2^n}$ are the value of the function at each n-tuple $\psi_{(i-1)}$, see 'Definitions' For example c_1 is the value of the function at n-tuple ψ_0 or when $x_1 = x_2 = \ldots x_n = 0$; c_2 is the value of the function at n-tuple ψ_1 or when $x_1 = x_2 = \ldots x_{n-1} = 0$, $x_n = 1$ etc .

Now it will be noted that the ordering of the coefficient vector is precisely that of the truth table representation of a Boolean function , -see 'Definitions'.

The example shown in Fig.1a. may therefore be written as :

$$\begin{array}{c} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \\ \mathbf{0} \; \mathbf{1} \; \mathbf{2} \; \mathbf{3} \; \mathbf{4} \; \mathbf{5} \; \mathbf{6} \; \mathbf{7} \; \mathbf{8} \; \mathbf{9} \; \mathbf{10} \; \mathbf{11} \; \mathbf{12} \; \mathbf{13} \; \mathbf{14} \; \mathbf{15} \end{array} \left.\begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix}\right] = U \Big]$$

where U is a Boolean function $F(x_1, x_2, x_3, x_4)$.

The numbers appearing below each member of the coefficient vector represent the n-tuples ψ_j, $0 \leqslant j \leqslant 2^n - 1$. Because the coefficient vector has a canonical form the ordering of these n-tuples is implied ; nevertheless it will be found convenient to include this information when the manipulation of matrices by paper-and-pencil methods is considered.

The representation of several Boolean functions is also possible , as in the case of conventional matrix algebra.

Consider

$$\begin{bmatrix} c_{1,u} & c_{2,u} & \cdot & \cdot & c_{2^n,u} \\ c_{1,v} & c_{2,v} & \cdot & \cdot & c_{2^n,v} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ c_{1,z} & c_{2,z} & \cdot & \cdot & c_{2^n,z} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} U \\ V \\ \cdot \\ \cdot \\ Z \end{bmatrix} \qquad \ldots (1.3)$$

which represents several n th order Boolean functions. In general the coefficient matrix will have p rows and 2^n columns, where p is the number of n th order Boolean functions to be represented.

As an example, the representation of three second order functions is given below.

The functions $U = x_1 \oplus x_2 \triangleq x_1 . \bar{x}_2 + \bar{x}_1 . x_2$,

$V = x_1 . x_2$,

$W = \bar{x}_1 + \bar{x}_2$

may be represented as $\begin{array}{c} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \\ \mathbf{0\ \ 1\ \ 2\ \ 3} \end{array} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} U \\ V \\ W \end{bmatrix}$.

In order that the values of a given set of functions may evaluated simultaneously for a particular n-tuple the column vector of the coefficient matrix corresponding to that n-tuple is extracted.

In the last example the values of the three functions corresponding to the n-tuple ψ_3 , where $x_1=1$ and $x_2=1$, is given by :

$$\begin{array}{c} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \\ \mathbf{0\ \ 1\ \ 2\ \ 3} \end{array} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

ie. the functions U,V,W have the values 0,1,0 respectively when $x_1=1$, $x_2=1$.

By re-writing the previous example with each n-tuple expanded as a Boolean matrix this process may be carried out by inspection :

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{Bmatrix} \quad \text{matrix form of n-tuples}$$

If the coefficient matrix is equal to the matrix of n-tuples the following matrix equation results :

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} U \\ V \end{bmatrix} \quad , \text{ for } n=2.$$

Clearly $\begin{bmatrix} U \\ V \end{bmatrix}$ will take the values of $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ over all n-tuples , ie.

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

; for this reason the matrix of n-tuples is called the Unit or Identity matrix and is denoted as $\left[A \right]$. The Unit matrix has , by definition, n rows and 2^n columns.

In general $$\left[A \right] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad \ldots \quad (1.4)$$

1.2.2 Matrix-Vector Multiplication.

It is now possible to mathematically define the operation which enables equations of the type

$$\left[C \right] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ \cdot \\ \cdot \\ \cdot \\ F_p \end{bmatrix} \qquad \ldots \quad (1.5)$$

to be evaluated. This operation will be termed matrix-vector multiplication.

Define: $\left[C \right]$ as a Boolean coefficient matrix having p rows and 2^n columns,

$\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$ as the defining variable vector having n rows,

$\begin{bmatrix} F_1 \\ F_2 \\ \cdot \\ \cdot \\ F_p \end{bmatrix}$ as the function vector having p rows and

$[A]$ as the matrix of n-tuples having n rows and 2^n columns.

The evaluation of equation (1.5) is then given by

$$\left. \begin{array}{l} F_i(x_1,x_2, \ldots ,x_n) = \bigcup\limits_{j=1}^{2^n} c_{i,j} \cap \left\{ \bigcap\limits_{k=1}^{n} (a_{k,j} \,\bar{\oplus}\, x_k) \right\} , \\ \qquad\qquad\qquad\qquad 1 \leqslant i \leqslant p \end{array} \right\} \ldots (1.6)$$

where $\oplus$ is the equivalence operator , $\bigcup$ represents union over a field , $\bigcap$ represents intersection over a field and $\cap$ represents intersection.

Equation (1.6) is interpreted in the following way :

$\bigcap\limits_{k=1}^{n} (a_{k,j} \oplus x_k)$ has the logical value 1 iff. the vector $\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$ is equal to the j th column of $[A]$. That is , the vector $\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$ is identified with the n-tuple corresponding to the j th column of $[A]$; this n-tuple is , by definition equal to ψ_{j-1}. Because no two n-tuples intersect in n-space this correspondence is unique.

Thence $\bigcup\limits_{j=1}^{2^n} c_{i,j} \cap \left\{ \bigcap\limits_{k=1}^{n} (a_{k,j} \oplus x_k) \right\}$ serves to extract the required member , row i column j , of $[C]$ corresponding to the function $F_i(x_1,x_2, \ldots ,x_n)$ and j th column of $[A]$.

The general expansion of equation (1.6) for n=2 is:

$$F_i(x_1,x_2) = c_{i,1}.(a_{1,1} \bar{\oplus} x_1).(a_{2,1} \bar{\oplus} x_2) + c_{i,2}.(a_{1,2} \bar{\oplus} x_1).(a_{2,2} \bar{\oplus} x_2)$$
$$+ c_{i,3}.(a_{1,3} \bar{\oplus} x_1).(a_{2,3} \bar{\oplus} x_2) + c_{i,4}.(a_{1,4} \bar{\oplus} x_1).(a_{2,4} \bar{\oplus} x_2)$$

1.2.3 Decimal Notation.

Boolean matrices and vectors may also be expressed in 'decimal notation' . An example of this notation has already been used to represent n-tuples. viz. ψ_j , $j = 2^{n-1} \times x_1 + 2^{n-2} \times x_2 + \ldots + 2^0 \times x_n$.

In general any Boolean matrix column vector may be expressed in decimal notation in the following way:

Let $\left[C\right]$ be a coefficient matrix having p rows and 2^n columns, then

$$c'_j = \sum_{k=1}^{p} 2^{n-p} \times c_{k,j} \quad , \qquad 1 \leqslant j \leqslant 2^n \qquad \ldots (1.7)$$

where c'_j is the j th column vector of $\left[C\right]$ expressed in decimal notation. The same technique can , of course , be applied to both vectors and matrices.

An example of the conversion of a Boolean matrix equation to decimal notation is :

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

, which may be expressed as

$$\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{Bmatrix}$$

$$\begin{array}{c} \begin{bmatrix} 1 & 5 & 5 & 2 \end{bmatrix} \\ 0 \;\; 1 \;\; 2 \;\; 3 \end{array} \begin{bmatrix} 3 \end{bmatrix} = \begin{bmatrix} 2 \end{bmatrix}$$

The unit matrix , by virtue of the fact that it is the matrix of n-tuples, may be defined in decimal notation as

$$a'_j \triangleq j-1 \quad , \quad 1 \leqslant j \leqslant 2^n \qquad \ldots (1.8)$$

where a'_j is the j th column vector of $\left[\underset{\sim}{A}\right]$ expressed in decimal notation.

The decimal notation is useful , not only as a shorthand method of expressing Boolean matrices , but also as a form which is convenient for the manipulation of such matrices by means of the digital computer.

1.2.4 Matrix-Network Topology.

A practical interpretation of Boolean matrix-vector multiplication is given in Fig.2. which corresponds to equation (1.5).

One of the most important properties of Boolean matrices is evident from this example , ie. it is possible to relate the row structure of a Boolean matrix equation to the topology of the logic circuit which it describes. The convention adopted here will be to relate the first row (function) of a coefficient matrix to the upper signal path at the output of the corresponding logic module , the second row of the coefficient matrix to the next-to-upper signal path at the output of the corresponding logic module , and so on. The same convention will be adopted for the defining variable vector and the corresponding logic module inputs.

1.2.5 Matrix Multiplication.

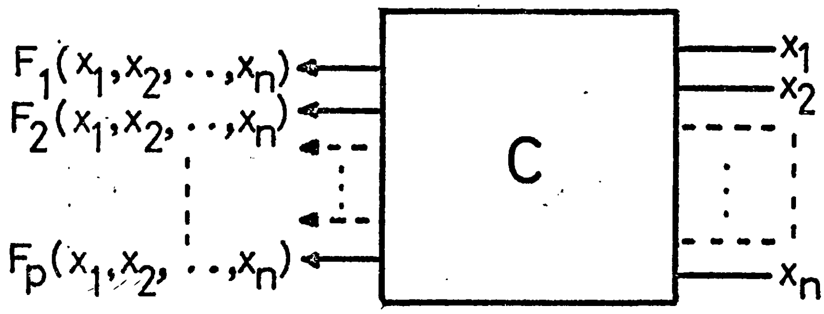
It is now possible to develop an operation termed 'Boolean matrix multiplication' which corresponds to the multiplication of conventional matrices.

Consider the identity $$[B][C]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = [D]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad \ldots (1.9)$$

and let $[B][C]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$ represent a pair of cascaded logic modules as shown in Fig.3. where the modules B and C correspond to $[B]$ and $[C]$ respectively. The dimensions of the matrices $[B]$, $[C]$ and $[D]$ follow from the discussion of the topological relationships above.

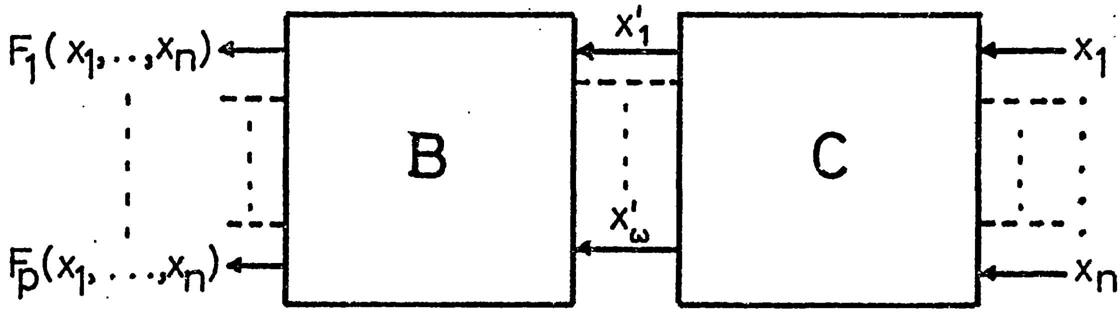
viz. $[C]$ will have ω rows and 2^n columns,

$[B]$ will have p rows and 2^ω columns



$$\begin{bmatrix} C \end{bmatrix}\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix} = \begin{bmatrix}F_1\\F_2\\\vdots\\F_p\end{bmatrix}$$

Fig. 2



$$\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} C \end{bmatrix}\begin{bmatrix}x_1\\\vdots\\x_n\end{bmatrix} = \begin{bmatrix}F_1\\\vdots\\F_p\end{bmatrix}$$

Fig. 3

and $[D]$ will have p rows and 2^n columns.

It should be noted that any deviation from this dimensioning results in a system which cannot be implémented.

A method of evaluating equation (1.9) is to first compute

$$[C]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ \cdot \\ \cdot \\ x'_\omega \end{bmatrix} \qquad \ldots (1.10)$$

Then equation (1.9) may be expressed as

$$[B]\begin{bmatrix} x'_1 \\ x'_2 \\ \cdot \\ \cdot \\ x'_\omega \end{bmatrix} = [D]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad \ldots (1.11)$$

Expressing equation (1.10) in the form given by equation (1.6) :

$$x'_i = \bigcup_{j=1}^{2^n} c_{i,j} \cap \left\{ \bigcap_{k=1}^{n} (a_{k,j} \;\bar{\oplus}\; x_k) \right\}, \qquad 1 \leqslant i \leqslant \omega \qquad \ldots (1.12)$$

Using the same method, equation (1.11) may be written as :

$$\left. \bigcup_{m=1}^{2^\omega} b_{r,m} \cap \left\{ \bigcap_{i=1}^{\omega} (a_{i,m} \;\bar{\oplus}\; x'_i) \right\} = \bigcup_{j=1}^{2^n} d_{r,j} \cap \left\{ \bigcap_{k=1}^{n} (a_{k,j} \;\bar{\oplus}\; x_k) \right\}, \quad 1 \leqslant r \leqslant p \right\} \ldots (1.13)$$

Now from equation (1.13)

$b_{r,m} = d_{r,j}$ when: $a_{k,j} = x_k$ and $a_{i,m} = x_i$;

and from equation (1.12)

$x'_i = c_{i,j}$ when: $a_{k,j} = x_k$,

whence

$d_{r,j} = b_{r,m}$ iff. $a_{i,m} = c_{i,j}$ and $a_{k,j} = x_k$.

The last equation is important because it enables the equation $[B][C] = [D]$ to be evaluated by again employing the general form of equation (1.6).

ie. given $d_{r,j} = b_{r,m}$ when $a_{i,m} = c_{i,j}$ then

$$d_{r,j} = \bigcup_{m=1}^{2^n} b_{r,m} \cap \left\{ \bigcap_{i=1}^{\omega} (a_{i,m} \;\bar{\oplus}\; c_{i,j}) \right\}, \quad 1 \leqslant r \leqslant p, \quad 1 \leqslant j \leqslant 2^n \quad \ldots (1.14)$$

Equation (1.14) may be interpreted using the same arguments applied to equation (1.6) : The j th column vector of $[C]$ is identified as the m th column vector of the unit matrix $[A]$; the j th column vector of $[D]$ must then be equal to the m th column vector of $[B]$.

An example of matrix multiplication is now given:

Evaluate $[B][C]\begin{bmatrix}x_1\\x_2\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\end{bmatrix}$ where $[B] = \begin{bmatrix}1&0&1&0\\0&0&1&1\\0&0&0&1\end{bmatrix}$,

and $[C] = \begin{bmatrix}1&0&0&1\\0&0&1&1\end{bmatrix}$.

For convenience the unit matrix , or matrix form of n-tuples,is written below $[B]$:

$$\begin{array}{c}\begin{bmatrix}1&0&1&0\\0&0&1&1\\0&0&0&1\end{bmatrix}\\ \begin{Bmatrix}0&0&1&1\\0&1&0&1\end{Bmatrix}\end{array}\begin{bmatrix}1&0&0&1\\0&0&1&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

Now the first column vector of $[C]$ corresponds to the third column vector of $[A]$ so that the first column vector of $[D]$ is equal to the third column vector of $[B]$. Similarly the second column vector of $[C]$ corresponds to the first column vector of $[A]$ so that the second column vector of $[D]$ is equal to the first column vector of $[B]$, and so forth. The complete solution together with the necessary operations can be shown as :

$$\begin{array}{c}\begin{bmatrix}1&0&1&0\\0&0&1&1\\0&0&0&1\end{bmatrix}\\ \begin{Bmatrix}0&0&1&1\\0&1&0&1\end{Bmatrix}\end{array}\begin{bmatrix}1&0&0&1\\0&0&1&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}1&1&0&0\\1&0&0&1\\0&0&0&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

The same equation expressed in decimal notation is :

$$\begin{array}{c}[4\ 0\ 6\ 3]\\ \mathbf{0\ 1\ 2\ 3}\end{array}[2\ 0\ 1\ 3]\begin{bmatrix}x_1\\x_2\end{bmatrix} = [6\ 4\ 0\ 3]\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

The implementation of this example is shown in Fig.4.

In general it can be shown that the operation of matrix multiplication is not commutative , ie.$[B][C] \neq [C][B]$. To show this consider the equations

$$[B][C] = [D] \qquad \ldots (1.15)$$

and

$$[C][B] = [D] \qquad \ldots (1.16)$$

Let $\langle B \rangle$, $\langle C \rangle$, $\langle D \rangle$ represent the sets of column vectors of $[B]$, $[C]$ and $[D]$ respectively.It is required to establish under which conditions equations (1.15) and (1.16) are simultaneously valid. From equations (1.14) and (1.15) a necessary condition is that

$$\langle D \rangle \subseteq \langle B \rangle \qquad \ldots (1.17)$$

and from equations(1.14) and (1.16) another necessary condition is that

$$\langle D \rangle \subseteq \langle C \rangle \qquad \ldots (1.18)$$

Equations (1.17) and (1.18) imply $\langle C \rangle \cap \langle B \rangle = \langle D \rangle$ which , in general,is not true.

One notable exception is $[A][C] = [C][A]$, where $[C]$ is any coefficient matrix and $[A]$ is the unit matrix.

It can be shown that the associative law holds however , eg. $[B]\big[[C][D]\big] = \big[[B][C]\big][D]$ etc.

1.2.6 <u>Basic Properties Reviewed</u>.

Several properties of the Boolean matrices and associated algebra are now noted.

1/ The algebra is similarly structured to that of conventional matrix algebra , having operations analogous to both vector-matrix and matrix-matrix multiplication.

2/ The structure of the matrices has the important property of defining logic modules not only in terms of functional behaviour but also in terms of input/output topology.

3/ The algebra is well suited to the description of multiple-output logic modules and may be used to evaluate the overall transfer function of cascades of such modules.

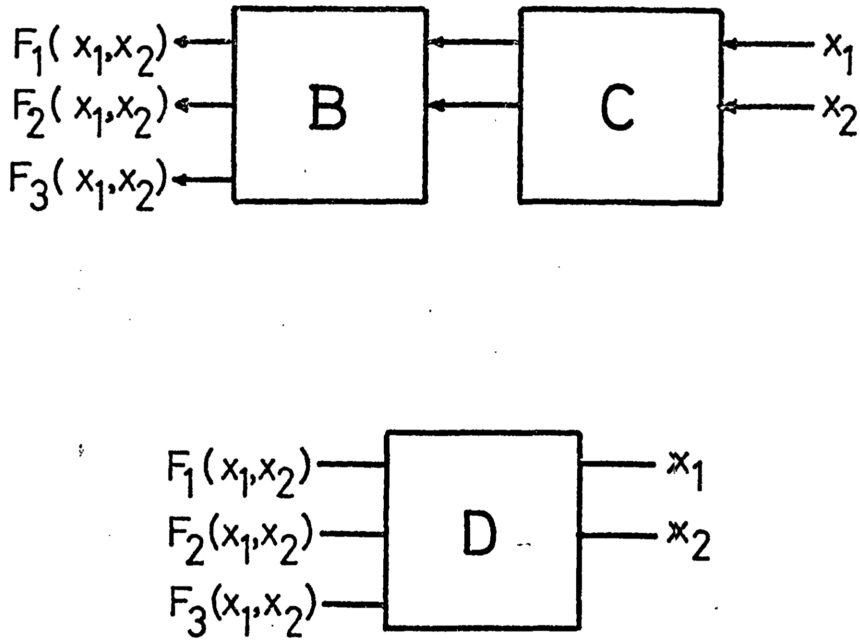

F1(x1,x2)
F2(x1,x2)
F3(x1,x2)
B
C
x1
x2
D

Fig. 4

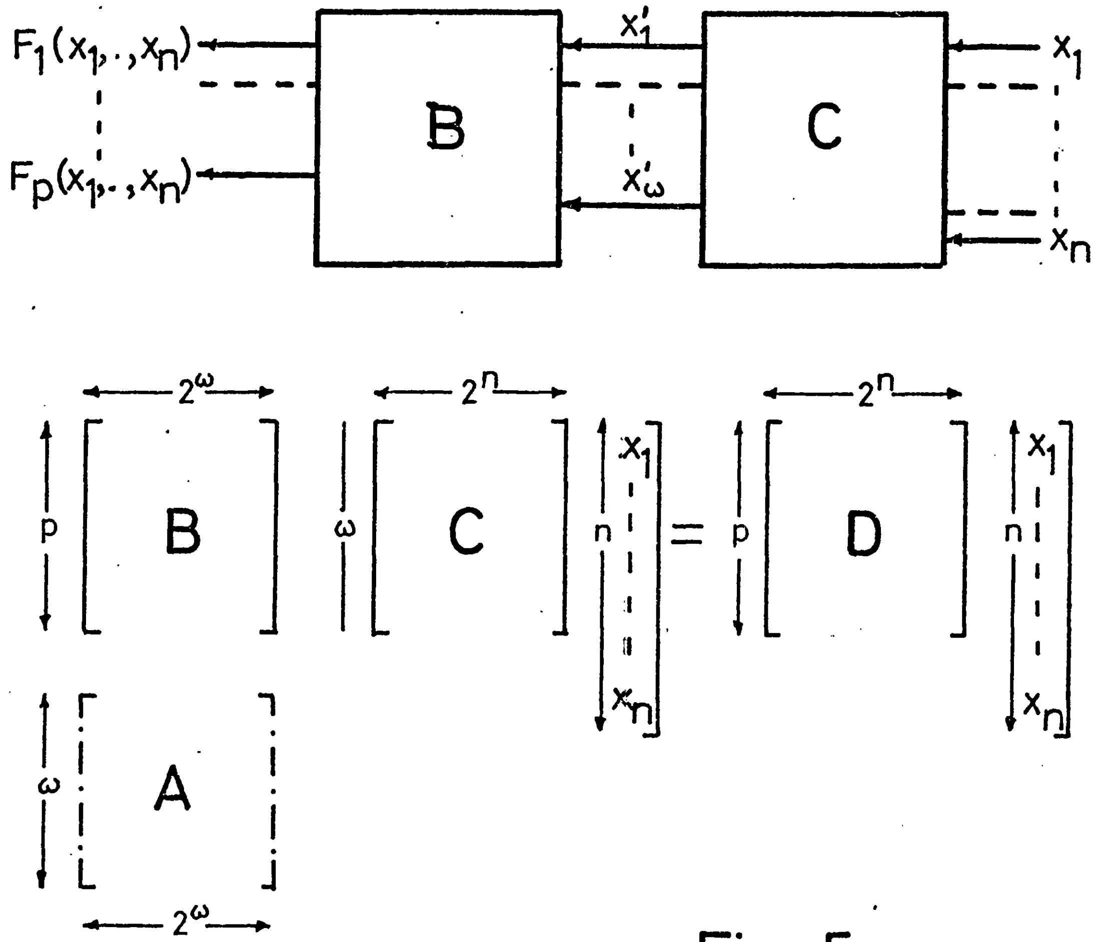

F1(x1,.,xn)
Fp(x1,.,xn)
B
C
x'1
x'ω
x1
xn
2ω
2n
p
ω
n
D
=
A

Fig. 5

4/ The matrices have a form well suited to manipulation by the digital computer.

Reference 5 should be consulted for further examples of the basic operations described in the previous sections

1.3 Further Properties.

1.3.1 Singular and Non-singular Matrices.

Before proceeding further it will be necessary to classify Boolean matrices into two categories , namely singular and non-singular.

A singular matrix is defined as a matrix having at least two column vectors identical.

A non-singular matrix is defined as a matrix having no column vectors identical - a special case is the unit matrix $[A]$.

An analogy can be drawn between the properties of singular/ non-singular matrices for both Boolean and conventional matrices as will be shown in the discussion of inverse matrices.

1.3.2 Dimensioning.

Consider the Boolean matrix equation

$$[B][C] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = [D] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} .$$

It is now convenient to investigate the relationships between the dimensions of the matrices $[B]$,$[C]$ and $[D]$.

Now the system under consideration has n defining variables ; therefore both $[C]$ and $[D]$ must have 2^n columns since they are defined on 2^n n-tuples ; see also equation (1.6). Suppose that $[C]$ has ω rows, ie. it describes a module with ω outputs . Then $[B]$ must be defined on ω inputs ; see also equation (1.14) . It follows that

$[B]$ has 2^{ω} columns. Now if $[D]$ has p rows, corresponding to p outputs, then $[B]$ also has p rows. These general dimensions are shown in Fig.5, p23.

In order that equations of the type discussed above may be solved given only the matrices $[B]$ and $[D]$ or $[C]$ and $[D]$ it is necessary to introduce the concept of the inverse matrix.

1.3.3 <u>The True Inverse</u>.

The inverse of a matrix, say $[C]$, is written as $[C]^{-1}$ and is defined by :

$$[C][C]^{-1} \triangleq [A] \triangleq [C]^{-1}[C] \qquad \ldots (1.19)$$

where $[A]$ is the unit matrix.

Let $[C]$ have ω rows and 2^n columns, then equation (1.19) is dimensioned as :

$$\underset{\omega}{\overset{\leftarrow 2^n \rightarrow}{[C]}}[C]^{-1} = [A] \qquad \ldots (1.20)$$

and

$$[C]^{-1}\,\underset{\omega}{\overset{\leftarrow 2^n \rightarrow}{[C]}} = [A] \qquad \ldots (1.21)$$

Equation (1.20) implies that $[A]$ has ω rows whilst equation (1.21) implies that $[A]$ has 2^n columns. The unit matrix A however, has n rows and 2^n columns by definition. It follows that $\omega = n$. In order that equation (1.19) shall hold therefore $[C]$ must have n rows and 2^n columns. Similarly $[C]^{-1}$ must have n rows and 2^n columns. Equation (1.19) is thus dimensioned :

$$\underset{n}{\overset{\leftarrow 2^n \rightarrow}{[C]}}\;\underset{n}{\overset{\leftarrow 2^n \rightarrow}{[C]^{-1}}} = \underset{n}{\overset{\leftarrow 2^n \rightarrow}{[A]}} = \underset{n}{\overset{\leftarrow 2^n \rightarrow}{[C]^{-1}}}\;\underset{n}{\overset{\leftarrow 2^n \rightarrow}{[C]}}$$

Now from the arguments used to develop equations (1.17) and (1.18) it follows that in equation (1.20) : $\langle A\rangle \subseteq \langle C\rangle$, and in equation (1.21) : $\langle A\rangle \subseteq \langle C^{-1}\rangle$, where $\langle A\rangle, \langle C\rangle, \langle C^{-1}\rangle$ represent the sets of column vectors of $[A], [C], [C]^{-1}$ respectively. Since $[A], [C], [C]^{-1}$ have the same dimensions and $[A]$ is non-singular then $\langle A\rangle = \langle C\rangle = \langle C^{-1}\rangle$ and both $[C]$ and $[C]^{-1}$ are non-singular.

Two necessary properties of inverse matrices are therefore :

1/ They are non-singular , as are the matrices from which they are derived.

2/ They have a row/column dimension ratio $n/2^n$, as do the matrices fron which they are derived.

These matrices will be termed 'true inverse matrices ' to distinguish them from other types of inverse matrices to be described later.

Now , by substitution in equation (1.14), equation (1.21) may be expressed as :

$$a_{r,j} = \bigcup_{m=1}^{2^n} c^{-1}_{r,m} \cap \left\{ \bigcap_{i=1}^{n} (a_{i,m} \bar{\oplus} c_{i,j}) \right\}, \quad 1 \leqslant r \leqslant n, \quad 1 \leqslant j \leqslant 2^n. \qquad \ldots (1.22)$$

That is $a_{r,j} = c^{-1}_{r,m}$ when $a_{i,m} = c_{i,j}$, which may be interpreted as follows :

If the j th column vector of $[C]$ is equal to the m th column vector of $[A]$ then the m th column vector of $[C]^{-1}$ is equal to the j th column vector of $[A]$.

Consider the following simple example :

Given $[C] = [2\ 3\ 1\ 0]$, $[A] \triangleq [0\ 1\ 2\ 3]$

construct $[C]^{-1}$.

Now the <u>first</u> column vector of $[C]$ is equal to the <u>third</u> column vector of $[A]$ so that the <u>third</u> column vector of $[C]^{-1}$ is equal to the <u>first</u> column vector of $[A]$, and so on.

This gives the result

$[C]^{-1} = [3\ 2\ 0\ 1]$, which may be verified from equation (1.20). viz.

$$[C][C]^{-1} = [A] \qquad \ldots ((1.20)\text{ repeated})$$

that is $[2\ 3\ 1\ 0][3\ 2\ 0\ 1] = [0\ 1\ 2\ 3]$

0 1 2 3

This procedure is readily implemented on the digital computer and may also be executed by inspection .

It is now possible to show that the equation

$$[B][C] = [D] \qquad \ldots (1.23)$$

is equivalent to $[B] = [D][C]^{-1}$ ... (1.24)

where $[C]$ is non-singular and has n rows and 2^n columns , whilst $[D]$ has rows and 2^n columns.ie.$[C]^{-1}$ is a true inverse.

Proof:

Using the general expression for matrix multiplication (eqn. (1.14)), equation (1.23) can be expressed as

$$d_{r,j} = \bigcup_{m=1}^{2^n} b_{r,m} \cap \left\{ \bigcap_{i=1}^{n} (a_{i,m} \bar{\oplus} c_{i,j}) \right\}, \quad 1 \leqslant r \leqslant \omega, \quad 1 \leqslant j \leqslant 2^n, \qquad \ldots (1.25)$$

and equation (1.24) can be expressed as

$$b_{r,m} = \bigcup_{m=1}^{2^n} d_{r,j} \cap \left\{ \bigcap_{i=1}^{n} (a_{i,j} \bar{\oplus} c^{-1}_{i,m}) \right\}, \quad 1 \leqslant r \leqslant \omega, \quad 1 \leqslant m \leqslant 2^n. \qquad \ldots (1.26)$$

Now from equation (1.25): $d_{r,j} = b_{r,m}$ when $a_{i,m} = c_{i,j}$,
and from equation (1.26) : $d_{r,j} = b_{r,m}$ when $a_{i,j} = c^{-1}_{i,m}$.
In order that equations (1.23) and (1.24) are equivalent it is therefore necessary that $a_{i,m} = c_{i,j}$ when $a_{i,j} = c^{-1}_{i,m}$. But this is exactly the condition which holds if $[C]^{-1}$ is a true inverse, as shown by equation (1.22).

Equation (1.23) is therefore equivalent to equation (1.24).

Q.E.D.

It can also be shown that equation (1.23) may be expressed as

$$[C] = [B]^{-1}[D] \qquad \ldots (1.27)$$

From equations (1.23),(1.24) and (1.27) it can be concluded that when a matrix equation is re-expressed in terms of the true inverses of its components , pre- and post-multiplicative ordering is preserved.

For example , in equation (1.23) the matrix $[C]$ post-multiplies $[B]$ and in equation (1.24) $[C]^{-1}$ post-multiplies $[D]$.

This is a property which is also found in conventional matrix algebra.

An example of the use of the true inverse matrix is now given :

A logic system is described by the equation

$$[B][C]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [D]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where $[C] = \begin{bmatrix} 0&1&0&1&1&0&1&0 \\ 0&1&1&0&0&1&1&0 \\ 1&1&0&1&0&1&0&0 \end{bmatrix}$ and $[D] = \begin{bmatrix} 0&1&1&0&1&0&0&1 \\ 0&1&1&1&1&0&1&0 \end{bmatrix}$.

Find the matrix $[B]$ (if it exists).

<u>Solution</u>

Convert the system equation into a form which enables $[B]$ to be evaluated :

$$[B] = [D][C]^{-1} ,$$

ie. $[C]^{-1}$ is required.

Inspection of $[C]$ shows it to have a row/column ratio of $n/2^n$ and in addition it is non-singular. $[C]^{-1}$ may therefore be evaluated.

Express $[C]$ in decimal notation and evaluate $[C]^{-1}$ from $[C]^{-1}[C] = [A]$ by inspection :

$$[C]^{-1}[1\ 7\ 2\ 5\ 4\ 3\ 6\ 0] = [0\ 1\ 2\ 3\ 4\ 5\ 6\ 7]$$

or

$$\begin{array}{c} [7\ 0\ 2\ 5\ 4\ 3\ 6\ 1] \\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7} \end{array}[1\ 7\ 2\ 5\ 4\ 3\ 6\ 0] = [0\ 1\ 2\ 3\ 4\ 5\ 6\ 7]$$

$$\text{ie.}\quad [C]^{-1} = [7\ 0\ 2\ 5\ 4\ 3\ 6\ 1]$$

Express $[D]$ in decimal notation and evaluate $[B]$ from $[B] = [D][C]^{-1}$:

$$[B] = \begin{array}{c} [0\ 3\ 3\ 1\ 3\ 0\ 1\ 2] \\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7} \end{array}[7\ 0\ 2\ 5\ 4\ 3\ 6\ 1] = [2\ 0\ 3\ 0\ 3\ 1\ 1\ 3]$$

This result can be checked by substitution in the original system equation.

The above example illustrates that the true inverse matrix may be used in logic synthesis. For example , in the above , $[D]$ may represent the transfer function of a required logic system and $[C]$ may represent an available logic module. The example shows that $[C]$ may be employed in the synthesis of $[D]$ giving a remaining module $[B]$ to be synthesised.

Of course it will be appreciated that in general the logic module corresponding to $[C]$ in the above example is not likely to have a transfer function described by a non-singular matrix having the correct dimensions which ensures the existance of a true inverse. The effect of relaxing the restrictions applied to the evaluation of inverse matrices is therefore considered below.

1.3.4 Valid Equations.

In order that criteria may be developed which allow the evaluation of the inverse of matrices not having the special properties necessary for the evaluation of the true inverse , it is first convenient to determine what constitutes a valid matrix equation.

Recalling the matrix equation

$$[B][C] = [D]$$ and the interpretation of equation (1.14).

viz. $d_{r,j} = b_{r,m}$ when $a_{i,m} = c_{i,j}$ (over the required limits),

the criteria which ensure the validity of the above matrix equation can be established.

It has already been established that one necessary condition that an equation of the above type shall be valid is that it has allowed dimensions. This will be assumed.

Consideration of the matrix [C] in the above shows that if two column vectors of [C] are identical then the two corresponding vectors of [D] must be identical.

ie. if $c_{i,j} = c_{i,k} = a_{i,m}$ then $d_{r,j} = d_{r,k} = b_{r,m}$.

However if two column vectors of [C] are different then the two corresponding vectors of [D] may or may not be different , depending upon the composition of [B].

ie. if $c_{i,j} = a_{i,m}$ and $c_{i,k} = a_{i,l}$ then $d_{r,j} = b_{r,m}$ and $d_{r,k} = b_{r,l}$ where $b_{r,m}$ may or may not be equal to $b_{r,l}$.

These observations give rise to :

Criterion 1.

A necessary condition that the matrix equation [B][C] = [D] shall be valid is that if [C] is singular then the identical column vectors of [C] shall correspond to the identical column vectors of [D] .

ooOoo

Consideration of the matrix [B] in the above equation shows that the set of unique column vectors of [D] must appear in the set of column vectors of [B] since $d_{r,j} = b_{r,m}$ when $a_{i,m} = c_{i,j}$.

It follows that [B] must have at least as many unique column vectors as there are unique column vectors in [D] . In addition [B] may be either singular or non-singular.

These observations give rise to :

Criterion 2.

A necessary condition that the matrix equation [B][C] = [D] shall be valid is that the set of unique column vectors of [D] shall appear in the set of column vectors of [B].

ooOoo

Now if either Criterion 1 or Criterion 2 is satisfied together with the dimensional restrictions, this is sufficient to guarantee the validity of a matrix equation of the type described above.

Specifically if , in the above equation, [C] and [D] are known and satisfy both the dimensional restrictions and Criterion 1 , then the matrix [B] may always be constructed. The same argument may be applied to the construction of [C] given [B] and [D] under Criterion 2 and the dimensional restrictions.

Since the matrices constructed under the above criteria may be singular or non-singular it follows that it should be possible to find the inverse of a singular matrix providing the result is only applied to valid matrix equations.

1.3.5 Inverse of Singular Matrices.

Let the inverse of a singular matrix be defined from :

$$\left[C\right]\left[C\right]^{-1} = \left[A\right] \qquad \ldots (1.28)$$

The evaluation of $\left[C\right]^{-1}$, where $\left[C\right]$ is singular is best illustrated by a simple example.

Suppose that $\left[C\right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$, or in decimal notation

$$\left[C\right] = \begin{bmatrix} 3 & 0 & 1 & 1 \end{bmatrix}$$

Substitution in equation (1.28) gives

$$\underset{\mathbf{0\ 1\ 2\ 3}}{\begin{bmatrix} 3 & 0 & 1 & 1 \end{bmatrix}} \left[C\right]^{-1} = \begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix}$$

Since $\left[C\right]$ has two rows it follows that $\left[A\right]$ has two rows and 2^2 columns, therefore $\left[C\right]^{-1}$ must have 2^2 columns and two rows.

Now by inspection it is clear that the first column vector of $\left[C\right]^{-1}$ must give rise to the value 0 , which is the first column vector of $\left[A\right]$, when $\left[C\right]^{-1}$ is multiplied by $\left[C\right]$. The only column vector of $\left[C\right]$ having a value 0 is that column vector corresponding to n-tuple 1. Consequently the first column vector of $[C]^{-1}$ must have the value 1.

The second column vector of $[C]^{-1}$ must give rise to the value 1 when $[C]^{-1}$ is multiplied by $[C]$. But $[C]$ has two column vectors with the value 1 , these appear at n-tuples 2 and 3 . The second column vector of $[C]^{-1}$ may therefore take the value 2 <u>or</u> 3.

The third column vector of $[C]^{-1}$ must give rise to the value 2 when $[C]^{-1}$ is multiplied by $[C]$. Now no column vector of value 2 appears in $[C]$ so that the third column vector of $[C]^{-1}$ is given the unspecified value '*' .

The fourth column vector of $[C]^{-1}$ must give rise to the value 3 when $[C]^{-1}$ is multiplied by $[C]$. Now $[C]$ has the value 3 only at n-tuple 0 ,consequently $[C]^{-1}$ must have its fourth column vector equal to 0.

This gives the result :

$$\begin{array}{c} \begin{bmatrix} 3 & 0 & 1 & 1 \end{bmatrix} \\ \mathbf{0\;\;1\;\;2\;\;3} \end{array} \begin{bmatrix} 1 & \substack{2\\3} & * & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix}$$

where
$$\begin{bmatrix} 1 & \substack{2\\3} & * & 0 \end{bmatrix} = [C]^{-1}$$

Now this inverse matrix may be employed in the evaluation of the following system :

<u>Given</u> $$[B][C]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [D]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

where $$[C] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

or in decimal notation $$[C] = \begin{bmatrix} 3 & 0 & 1 & 1 \end{bmatrix}$$

and $$[D] = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

or in decimal notation $$[D] = \begin{bmatrix} 1 & 3 & 2 & 2 \end{bmatrix}$$

evaluate $[B]$.

Solution

Substitution in the equation $[B][C] = [D]$ gives

$$[B]\begin{bmatrix}3 & 0 & 1 & 1\end{bmatrix} = \begin{bmatrix}1 & 3 & 2 & 2\end{bmatrix}$$

This equation satisfies Criterion 1 and is dimensionally correct. $[B]$ may therefore be evaluated from $[B] = [D][C]^{-1}$.

Now $[C]^{-1}$ has been evaluated as $\begin{bmatrix}1 & \frac{2}{3} & * & 0\end{bmatrix}$. Substitution in the above equation gives

$$[B] = \begin{array}{c}\begin{bmatrix}1 & 3 & 2 & 2\end{bmatrix}\\ \begin{matrix}\mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3}\end{matrix}\end{array}\begin{bmatrix}1 & \frac{2}{3} & * & 0\end{bmatrix}$$

$$= \begin{bmatrix}3 & 2 & * & 1\end{bmatrix}$$

This result may be checked by substitution in the given equation :

$$\begin{array}{c}\begin{bmatrix}3 & 2 & * & 1\end{bmatrix}\\ \begin{matrix}\mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3}\end{matrix}\end{array}\begin{bmatrix}3 & 0 & 1 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}1 & 3 & 2 & 2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Note that the symbol '*' is used to indicate that the column vector may take any value. This must be so in the above equation since the relevant column vector is not involved when $[C]$ is multiplied by $[B]$. However, in order that $[B]$ shall represent a real system, the value of '*' must lie within the dimensional restrictions of $[B]$.

Matrices having column vectors with more than one possible value will be termed 'multi-valued'.

The fact that the singular inverse of a matrix may always be used to solve matrix equations which are valid under Criteria 1 and 2 together with the dimensional restrictions can be proved using methods similar to those applied to equations (1.22),(1.25) and (1.26) .

In the previous example the inverted matrix had a row/column dimension ratio of $n/2^n$, but this is not a necessary condition for the evaluation of inverse matrices as is illustrated by the following example.

<u>Given</u> $$[B][C]\begin{bmatrix}x_1\\x_2\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\end{bmatrix} ,$$

where $$[B] = \begin{bmatrix}0&1&1&1\\1&0&1&0\\1&1&0&1\end{bmatrix}$$ or in decimal notation $$= \begin{bmatrix}3&5&6&5\end{bmatrix}$$

and $$[D] = \begin{bmatrix}1&0&0&1\\1&1&1&0\\0&1&1&1\end{bmatrix}$$ or in decimal notation $$= \begin{bmatrix}6&3&3&5\end{bmatrix}$$

find $[C]$.

<u>Solution</u>

Both $[B]$ and $[D]$ have 3 rows , the equation therefore has the correct dimensions.

The set of unique column vectors of $[D]$ are $\langle 3,5,6\rangle$ which appear in $[B]$.

The equation is therefore dimensionally correct and satisfies Criterion 2 , it is thus a valid equation.

Evaluate $[B]^{-1}$ from $[B][B]^{-1} = [A]$ by inspection :

$$\begin{array}{c}\begin{bmatrix}3&5&6&5\end{bmatrix}\\ \begin{matrix}0&1&2&3\end{matrix}\end{array}\begin{bmatrix}*&*&*&0&*&\frac{1}{3}&2&*\end{bmatrix} = \begin{bmatrix}0&1&2&3&4&5&6&7\end{bmatrix}$$

Note that $[B]$ has 3 rows therefore $[A]$ has three rows and 2^3 columns . Then $[B]^{-1}$ has 2^3 columns and 2 rows.

Find $[C]$ from $[C] = [B]^{-1}[D]$:

$$[C] = \begin{array}{c}\begin{bmatrix}*&*&*&0&*&\frac{1}{3}&2&*\end{bmatrix}\\ \begin{matrix}0&1&2&3&4&5&6&7\end{matrix}\end{array}\begin{bmatrix}6&3&3&5\end{bmatrix}$$

$$= \begin{bmatrix}2&0&0&\frac{1}{3}\end{bmatrix}$$

This result may be checked by substitution in the given equation :

$$\begin{array}{c}\begin{bmatrix}3&5&6&5\end{bmatrix}\\ \begin{matrix}0&1&2&3\end{matrix}\end{array}\begin{bmatrix}2&0&0&\frac{1}{3}\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}6&3&3&5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

1.3.6 Multi-valued Matrices.

The study of the composition of inverse singular matrices has resulted in the consideration of multi-valued matrices. It is of interest to consider the more general aspects of multi-valued matrices in order that systems specified with 'don't care' conditions may be manipulated.

Consider the following equation :

$$\begin{array}{c} \begin{bmatrix} 1 & 0 & * & 1 & 1 & 0 & 1 & * \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ * & 1 & * & 0 & 0 & 1 & * & 0 \end{bmatrix} \\ \begin{matrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \end{matrix} \end{array} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

, where '*' denotes a don't care condition (0 or 1). For example $F_1(x_1,x_2,x_3)$ may take the value 0 or 1 at n-tuples 2 and 7 .

In decimal notation this equation may be written as

$$\begin{array}{c} \begin{bmatrix} \frac{4}{5} & 1 & \begin{smallmatrix} 2 \\ 3 \\ 6 \\ 7 \end{smallmatrix} & 6 & 6 & 3 & \frac{6}{7} & \frac{2}{6} \end{bmatrix} \\ \begin{matrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \end{matrix} \end{array} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

since ,for example, the column vector at n-tuple 2 may take any of the values $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

For the general equation $\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix} D \end{bmatrix}$ it has been shown that $d_{r,j} = b_{r,m}$ when $a_{i,m} = c_{i,j}$ (over the allowed dimensional limits)

Now suppose that $\begin{bmatrix} B \end{bmatrix}$ is multi-valued where $b_{r,m}$ has either the value α or β , then $d_{r,j}$ will also take the value α or β when $a_{i,m} = c_{i,j}$.

Similarly if $\begin{bmatrix} C \end{bmatrix}$ is multi-valued where $c_{i,j} = a_{i,m}$ or $c_{i,j} = a_{i,l}$ then $d_{r,j}$ will take the values $b_{r,m}$ or $b_{r,l}$.

It is therefore possible to apply the methods of Boolean matrix algebra to general multi-valued matrices without recourse to special techniques.

An important property of multi-valued matrices is that it is possible to use them to define relationships between functions.

Consider the following equation :

$$\begin{bmatrix} 4 & \begin{smallmatrix}3\\6\end{smallmatrix} & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

which may be written as either

$$\begin{array}{c} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \\ \mathbf{0\ \ 1\ \ 2\ \ 3} \end{array} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

or

$$\begin{array}{c} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{0\ \ 1\ \ 2\ \ 3} \end{array} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

Inspection of the last two equations shows that the function $F_1(x_1,x_2)$ is related to the function $F_3(x_1,x_2)$. Specifically , at n-tuple 1 , $F_1(0,1)$ has the value 0 only if $F_2(0,1)$ has the value 1 and vice-versa .

The given equation therefore defines two dependent functions and in this respect differs from the type of multi-valued matrix considered so far.

1.3.7 <u>Conditionally and Unconditionally valid equations</u>.

Some care must be taken when manipulating multi-valued matrices to establish the correct interpretation of the functions they represent.

Consider the following equation

$$\begin{bmatrix} B \end{bmatrix} \begin{bmatrix} 2 & 1 & * & \frac{1}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 3 & * & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Now this equation may be written as

$$\begin{bmatrix} B \end{bmatrix} \begin{bmatrix} 2 & 1 & * & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 3 & * & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

or

$$\begin{bmatrix} B \end{bmatrix} \begin{bmatrix} 2 & 1 & * & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 3 & * & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} .$$

In either case a valid equation is formed under Criterion 1 .

Such an equation is termed'unconditionally valid'.

The matrix $\begin{bmatrix} B \end{bmatrix}$ may then be evaluated in the following way :

For the first form of the equation $[B] = [1\ 3\ *\ 3][2\ 1\ *\ 1]^{-1}$

and for the second form $[B] = [1\ 3\ *\ 3][2\ 1\ *\ 3]^{-1}$.

Computing the required inverses from $[C][C]^{-1} = [A]$ by inspection : firstly $\begin{array}{c}[2\ 1\ *\ 1]\\ \mathbf{0\ 1\ 2\ 3}\end{array}[*\ \frac{1}{3}\ 0\ *] = [0\ 1\ 2\ 3]$

and secondly $\begin{array}{c}[2\ 1\ *\ 3]\\ \mathbf{0\ 1\ 2\ 3}\end{array}[*\ 1\ 0\ 3] = [0\ 1\ 2\ 3]$.

ie. $[2\ 1\ *\ 1]^{-1} = [*\ \frac{1}{3}\ 0\ *]$ and $[2\ 1\ *\ 3]^{-1} = [*\ 1\ 0\ 3]$

Evaluating $[B]$ from $[B] = [D][C]^{-1}$ for both cases :

$$[B] = \begin{array}{c}[1\ 3\ *\ 3]\\ \mathbf{0\ 1\ 2\ 3}\end{array}[*\ \tfrac{1}{3}\ 0\ *]$$

$$= [*\ 3\ 1\ *]$$

or $$[B] = [*\ 3\ 1\ 3]$$

Now for both forms of $[B]$ to satisfy the original equation the fourth column vector of $[B]$ must satisfy both values '*' and '3' . Since '*' represents an unspecified vector which includes the value '3' the fourth column vector of $[B]$ must be constrained to take the value '3' .

The original equation can therefore be written as :

$$\begin{array}{c}[*\ 3\ 1\ 3]\\ \mathbf{0\ 1\ 2\ 3}\end{array}[2\ 1\ *\ \tfrac{1}{3}]\begin{matrix}x_1\\x_2\end{matrix}\Bigg] = [1\ 3\ *\ 3]\begin{matrix}x_1\\x_2\end{matrix}\Bigg]$$

which may be checked by inspection.

Not all equations are unconditionally valid however. Consider the following equation :

$$[B][2\ 0\ *\ \tfrac{0}{2}]\begin{matrix}x_1\\x_2\end{matrix}\Bigg] = [1\ 3\ *\ \tfrac{1}{3}]\begin{matrix}x_1\\x_2\end{matrix}\Bigg] .$$

This equation may take any of the following four forms :

$$[B][2\ 0\ *\ 0]\begin{matrix}x_1\\x_2\end{matrix}\Bigg] = [1\ 3\ *\ 1]\begin{matrix}x_1\\x_2\end{matrix}\Bigg]$$

$$[B][2\ 0\ *\ 0]\begin{matrix}x_1\\x_2\end{matrix}\Bigg] = [1\ 3\ *\ 3]\begin{matrix}x_1\\x_2\end{matrix}\Bigg]$$

$$[B][2\ 0\ *\ 2]\begin{matrix}x_1\\x_2\end{matrix}\Bigg] = [1\ 3\ *\ 1]\begin{matrix}x_1\\x_2\end{matrix}\Bigg]$$

$$[B]\,[2\ 0\ *\ 2]\begin{bmatrix}x_1\\x_2\end{bmatrix} = [1\ 3\ *\ 3]\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

The application of Criterion 1 shows that only the second and third forms of this equation are valid.

This equation will be written as

$$[B]\,[2\ 0\ *\ \substack{0\\2}]\begin{bmatrix}x_1\\x_2\end{bmatrix} = [1\ 3\ *\ \substack{3\\1}]\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

where the tie symbol is used to indicate that certain multi-valued column vectors are related. The expression above indicates that '0' in one matrix implies '3' in the other whilst '2' in the first matrix implies '1' in the second.

Matrix equations of this type will be called 'conditionally valid' .

In the above example the matrix $[B]$ may be evaluated (for the valid forms of the equation) using the same method described in the previous example. This allows the original equation to be written as :

$$\begin{array}{cccc}\multicolumn{4}{c}{[3\ *\ 1\ *]}\\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3}\end{array}[2\ 0\ *\ \substack{0\\2}]\begin{bmatrix}x_1\\x_2\end{bmatrix} = [1\ 3\ *\ \substack{3\\1}]\begin{bmatrix}x_1\\x_2\end{bmatrix} .$$

Another example is as follows :

$$[B]\,[3\ 7\ \substack{\binom{4}{6}\\3}\ 7]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = [2\ 3\ \substack{(5)\\2}\ 3]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$$

, where the values '4' or '6' in the first matrix are related to the value '5' in the second matrix . Also the value '3' in the first matrix is related to the value '2' in the second matrix.. This gives the result :

$$\begin{array}{cccccccc}\multicolumn{8}{c}{[*\ *\ *\ 2\ 5\ *\ 5\ 3]}\\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7}\end{array}[3\ 7\ \substack{\binom{4}{6}\\3}\ 7]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = [2\ 3\ \substack{(5)\\2}\ 3]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} .$$

1.3.8 Matrices Raised to Exponents.

It has been shown that the multiplication of two Boolean matrices can be interpreted as a cascade of two logic modules. Consider the case where the two logic modules are identical, each being represented by the matrix $\left[C\right]$. The overall transfer function of the system is then given by

$$\left[C\right]\left[C\right]\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} F_1 \\ \cdot \\ \cdot \\ F_n \end{bmatrix}$$

For convenience this equation may be written in the form

$$\left[C\right]^2\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} F_1 \\ \cdot \\ \cdot \\ F_n \end{bmatrix}$$

In practical terms it is clear that $\left[C\right]$ must have a row/column ratio of $n/2^n$. If this were not so a situation would arise where the number of outputs from one module would differ from the number of inputs to the next, which is topologically inconsistant. This also means that the number of functions generated by the cascade is n. See also Fig.6.

In general π such cascaded modules may be represented by :

$$\left[C\right]^{\pi}\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} F_1 \\ \cdot \\ \cdot \\ F_n \end{bmatrix} \qquad \ldots (1.29)$$

The expression $\left[C\right]$ in the above will be refered to as raising the matrix $\left[C\right]$ to the power π. ie π is an exponent.

Consider the effect of raisingthe following non-singular matrix in power.

$$\left[C\right]^1 = \left[3\ 0\ 1\ 2\right]$$
$$\left[C\right]^2 = \left[3\ 0\ 1\ 2\right]\left[3\ 0\ 1\ 2\right] = \left[2\ 3\ 0\ 1\right]$$
$$\left[C\right]^3 = \left[C\right]^2\left[C\right] = \left[2\ 3\ 0\ 1\right]\left[3\ 0\ 1\ 2\right] = \left[1\ 2\ 3\ 0\right]$$

etc.

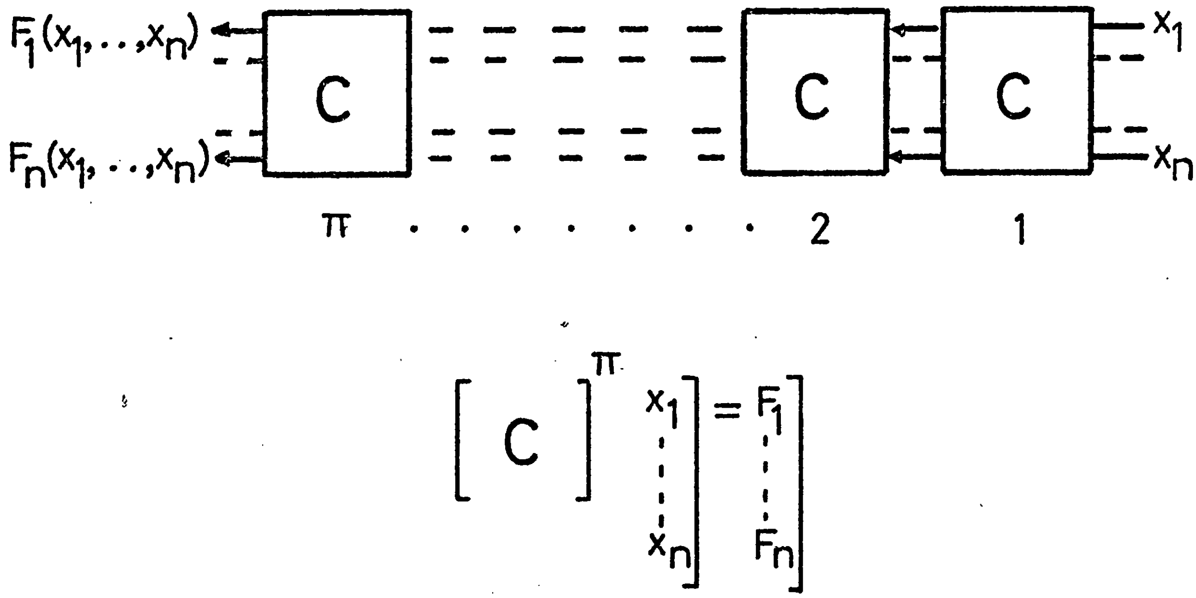

$F_1(x_1,\ldots,x_n)$
$F_n(x_1,\ldots,x_n)$
C
C
C
x_1
x_n
π
2
1
$[\,C\,]^{\pi}\begin{bmatrix}x_1\\ \vdots\\ x_n\end{bmatrix}=\begin{bmatrix}F_1\\ \vdots\\ F_n\end{bmatrix}$

Fig.6

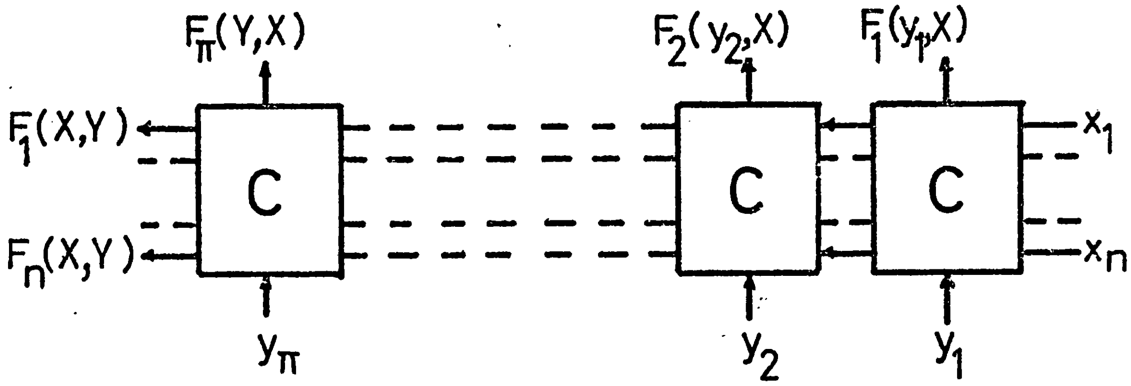

$F_\pi(Y,X)$
$F_2(y_2,X)$
$F_1(y_1,X)$
$F_1(X,Y)$
$F_n(X,Y)$
C
C
C
x_1
x_n
y_π
y_2
y_1

Fig.7

For convenience this process may be expressed by means of a <u>power table</u> :

MATRIX	π
0 1 2 3	0
3 0 1 2	1
2 3 0 1	2
1 2 3 0	3
0 1 2 3	4
3 0 1 2	5
2 3 0 1	6
.	.
.	.
.	.
.	.

where any matrix raised to a zero exponent is defined as the unit matrix.

From the previous discussion of true inverse matrices it is possible to construct the negative part of the power table for the above example using the definition

$$[C]^{-\pi} = \left[[C]^{\pi}\right]^{-1} \qquad \ldots (1.30)$$

The complete table then becomes :

MATRIX	π
.	.
.	.
.	.
1 2 3 0	-5
0 1 2 3	-4
3 0 1 2	-3
2 3 0 1	-2
1 2 3 0	-1
0 1 2 3	0
3 0 1 2	1
2 3 0 1	2
1 2 3 0	3
0 1 2 3	4
3 0 1 2	5
.	.
.	.
.	.

Now it can be shown that the additive law of indices holds for this algebra.

Consider the equation $[C]^{P}\,[C]^{-Q} = [R]$

which may be expanded as :

$$\Big[[C]_1\,[C]_2 \;\ldots\; [C]_P\Big]\Big[[C]^{-1}_1\,[C]^{-1}_2 \;\ldots\; [C]^{-1}_Q\Big] = [R]$$

Using the relationship $[C][C]^{-1} = [A]$ this equation may be expressed as ;

$$\Big[[C]_1\,[C]_2 \;\ldots\; [C]_{P-1}\Big]\,[A]\,\Big[[C]^{-1}_2\,[C]^{-1}_3 \;\ldots\; [C]^{-1}_Q\Big] = [R]$$

or $$[C]^{P-1}\,[C]^{-(Q-1)} = [R]$$

After applying this technique P times the following result is obtained :

$$[C]^{0}\,[C]^{-(Q-P)} = [R]$$

or $$[C]^{P-Q} = [R]$$

This gives the result

$$[C]^{P}\,[C]^{-Q} = [C]^{P-Q} \qquad \ldots (1.31)$$

In the previous power table for example

$$[C]^2\,[C]^{-3} = \begin{bmatrix} 2 & 3 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} \end{bmatrix}\begin{bmatrix} 3 & 0 & 1 & 2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 2 & 3 & 0 \end{bmatrix}$$
$$= [C]^{-1}$$

Another example of a power table is as follows

MATRIX	π
.	.
.	.
2 1 0 3	-3
0 1 2 3	-2
2 1 0 3	-1
0 1 2 3	0
2 1 0 3	1
0 1 2 3	2
2 1 0 3	3
.	.
.	.

In this example the value of the matrix at n-tuples 1 and 3 remain the same when the matrix is raised in power to any positive or negative exponent. Such column vectors , which in the defining matrix have the property of being identical to the n-tuples on which they are defined , will be called eigenvectors .

If a cascade of the type shown in Fig. 6 has a set of inputs corresponding to an eigenvector then it follows that the outputs of each of the cascaded modules will also have that value.

eg. in the previous example

$$\begin{bmatrix}2 & 1 & 0 & 3\end{bmatrix}^{\pi} \begin{bmatrix}1\end{bmatrix} = \begin{bmatrix}1\end{bmatrix}$$

or $$\begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 0 & 1\end{bmatrix}^{\pi} \begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}0\\ 1\end{bmatrix}$$ for all values of π .

Now Hennie , see reference 6 , has shown that such cascades may be considered as transformed finite-state machines. If such a machine is started in a state corresponding to an eigenvector then it will remain in that state.

It is of theoretical interest to note that the algebra upon which a power table is constructed forms a group with Boolean matrix multiplication as the group operation. A defining matrix then forms the generator for a sub-group. Because these sub-groups have a single generator they are cyclic. This is evident from the examples of power tables so far considered. Such cyclic groups are abelian, ie. for any two members of the group a,b , $a*b = b*a$ where $*$ denotes the group operation.

A power table may also be constructed with a singular matrix as a generator but the group properties mentioned above no longer hold.

Consider the following table :

MATRIX	π
$\vdots$	
$0\ *\ *\ \begin{smallmatrix}1\\2\\3\end{smallmatrix}$	-3
$0\ *\ *\ \begin{smallmatrix}1\\2\\3\end{smallmatrix}$	-2
$0\ *\ 1\ \begin{smallmatrix}2\\3\end{smallmatrix}$	-1
0 1 2 3	0
0 2 3 3	1
0 3 3 3	2
0 3 3 3	3
$\vdots$	

which has an eigenvector '0'. The inverses of the singular matrices have been computed using the methods previously described.

These singular inverse matrices may only be employed in the solution of valid matrix equations. For example equations of the type

$$[B]\,[C]^{\pi}\begin{bmatrix}x_1\\ \vdots\\ x_n\end{bmatrix} = [D]\begin{bmatrix}x_1\\ \vdots\\ x_n\end{bmatrix}$$

to be evaluated for $[B]$, if they are valid , using the identity $[B] = [D][C]^{-\pi}$.

The law of the addition of indices must be applied with great care to such tables as is shown by the following example.

Suppose in the above power table only $[C]^2$ and $[C]^3$ are known. It is required to evaluate $[C]$.

Two identities may be established immediately , namely

$$[C]^2\,[C] = [C]^3$$

and

$$[C]\,[C]^2 = [C]^3 .$$

For the first identity $[C] = [C]^{-2}\,[C]^3$

$$= \left[0\ *\ *\ \begin{smallmatrix}1\\2\\3\end{smallmatrix}\right]\left[0\ 3\ 3\ 3\right]$$

0 1 2 3

then $\quad [C] = \left[0 \ \substack{1\\2\\3} \ \substack{1\\2\\3} \ \substack{1\\2\\3}\right]$

and for the second identity $\quad [C] = [C]^3\,[C]^{-2}$

$$= \begin{array}{c}\left[0 \ 3 \ 3 \ 3\right] \\ \mathbf{0\ 1\ 2\ 3}\end{array}\left[0 \ * \ * \ \substack{1\\2\\3}\right]$$

$$= [0 \ * \ * \ 3].$$

Because a cascade of identical modules is under consideration it is known that these two forms of $[C]$ are compatible. For $[C]$ to lie within these restrictions it must have the form

$$[C] = \begin{array}{c}\left[0 \ \substack{1\\2\\3} \ \substack{1\\2\\3} \ 3\right] \\ \mathbf{0\ 1\ 2\ 3}\end{array}$$

Now $[C]^2$ is known to have the value $[0 \ 3 \ 3 \ 3]$ it follows therefore that $[C]$ cannot have eigenvectors at n-tuples 1 and 2 . This reduces the possible form of $[C]$ to : $\quad [C] = \left[0 \ \substack{2\\3} \ \substack{1\\3} \ 3\right]$

Finally each possible form of $[C]$ is squared

$$[0 \ 2 \ 1 \ 3]^2 = [0 \ 1 \ 2 \ 3]$$
$$[0 \ 2 \ 3 \ 3]^2 = [0 \ 3 \ 3 \ 3]$$
$$[0 \ 3 \ 1 \ 3]^2 = [0 \ 3 \ 3 \ 3]$$
$$[0 \ 3 \ 3 \ 3]^2 = [0 \ 3 \ 3 \ 3] \quad .$$

The first result does not satisfy the known result for $[C]^2$ so that $[C]$ must have a conditional form :

$$[C] = \left[0 \ \overset{\frown}{\substack{\left(\substack{2\\3}\right)\\3} \ \substack{(3)\\1}} \ 3\right]$$ where the tie symbol has the usual meaning.

Note that the value of $[C]$ actually used to generate the power table falls within this definition.

If the two possible forms of $[C]$ are now expanded :

$$[C] = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & * & 1 & 1 \end{bmatrix}$$

or $$[C] = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

it is evident that no dependent functions are involved.

These techniques may also be applied to circuits of the type shown in Fig .7, see Section 1.5.5.

1.3.9 Matrix Root Extraction.

It is of interest to be able to extract the roots of a given matrix in order that a particular system may be synthesised as a cascade of identical logic modules or 'iterative cascade ', see references 6 and 7 .

The R th root of a matrix $[C]$ will be written as $[C]^{\frac{1}{R}}$ and defined by

$$\left[[C]^{\frac{1}{R}}\right]^{R} = [C] \qquad \ldots (1.32)$$

It can be shown that this equation is , in general , non-linear and thus cannot be solved by classical methods.

A special case where root extraction is possible is when the given matrix generates a cyclic power table. In such a table it is always true that

$$[C]^{(1+k\eta)} = [C] \qquad \ldots (1.33)$$

where the matrix $[C]$ appears cyclically in the table at intervals of power η .(k is any positive integer.)

It is then true that

$$[C]^{\frac{1}{1+k\eta}} = [C] \qquad \ldots (1.34a)$$

and

$$[C]^{-\left(\frac{1}{1+k\eta}\right)} = [C]^{-1} \qquad \ldots (1.34b)$$

which enables certain roots to be evaluated.

Consider the following example :

MATRIX	π
.	
.	
.	
0 4 $\frac{1}{3}$ * 7 2 6 5	-6
0 $\frac{1}{3}$ 2 * 4 5 6 7	-5
0 2 5 * $\frac{1}{3}$ 7 6 4	-4
0 5 7 * 2 4 6 $\frac{1}{3}$	-3
0 7 4 * 5 $\frac{1}{3}$ 6 2	-2
0 4 $\frac{1}{3}$ * 7 2 6 5	-1
0 1 2 3 4 5 6 7	0
0 2 5 2 1 7 6 4	1
0 5 7 5 2 4 6 1	2
0 7 4 7 5 1 6 2	3
0 4 1 4 7 2 6 5	4
0 1 2 1 4 5 6 7	5
0 2 5 2 1 7 6 4	6
.	
.	
.	

Here $\eta=5$ whence from equation(1.33) $[C]^{(1+5k)}=[C]$. For k=1 and from equation (1.34a) $[C]^{\frac{1}{6}}=[C]$. Squaring both sides of this expression gives $[C]^{\frac{1}{3}}=[C]^2$, that is the cube root of $[C]$ is equal to $[C]^2$, or

$$[C]^{\frac{1}{3}} = [0\ 5\ 7\ 5\ 2\ 4\ 6\ 1] .$$

Unfortunately the generation of a cyclic table represents a special case.

For the general case the following points are noted :

1/ There are cases where no specific roots of a given matrix can be found , and there are cases where more than one root can be found.

2/ If $[C]$ is non-singular then $[C]^{\frac{1}{R}}$ is non-singular and if $[C]$ is singular then $[C]^{\frac{1}{R}}$ is singular.

3/ If $[C]$ has no eigenvectors then $[C]^{\frac{1}{R}}$ has no eigenvectors.

4/ In general the logic modules corresponding to the roots of a system are of comparable complexity to the logic module which will synthesise the overall system.

5/ The number of functions synthesisable in terms of cellular cascades as a proportion of the total number of possible functions becomes small as n becomes large. See reference 7 pp. 105-161 .

1.4 Boolean Matrix Operators.

1.4.1 Post-multiplicative Operators.

Consider the matrix equation

$$[C][\emptyset]\begin{bmatrix}x_1\\x_2\\\cdot\\\cdot\\x_n\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\\\cdot\\\cdot\\x_n\end{bmatrix} \quad .$$

Now $[\emptyset]$ post-multiplies $[C]$ and will be called a post-multiplicative operator.

From equation (1.14)

$$d_{r,j} = c_{r,m} \quad \text{when} \quad a_{i,m} = \emptyset_{i,j} \quad \text{(over the allowed limits)}$$

Clearly the matrix $[D]$ is composed of certain column vectors of $[C]$ which have been perturbed in n-space according to the composition of the column vectors of $[\emptyset]$.

Suppose that $[\emptyset]$ is non-singular and has n rows and 2^n columns , then $\langle D\rangle \subseteq \langle C\rangle$ where $\langle D\rangle$ and $\langle C\rangle$ represent the sets of column vectors of $[D]$ and $[C]$ respectively . Because $[\emptyset]$ is non-singular $[D]$ will be composed of a permutation of the column vectors of $[C]$. That is the functions represented by $[D]$ are those functions represented by $[C]$ but permuted in n-space ; no information about the functions of $[C]$ is lost ; they are re-constructable from $[D]$.

Some special forms of $[\emptyset]$ will now be considered.

Suppose that $[\emptyset]$ is equal to the unit matrix $[A]$.

Then $[C][\emptyset]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix} = [C]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix}$.

Consider now the effect of making $[\emptyset]$ identical to $[A]$ except that the h th row of $[\emptyset]$ is equal to the complement of the h th row of $[A]$. Then applying equation (1.6) to $[\emptyset]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix} = \begin{bmatrix}F_1\\F_2\\ \cdot\\ \cdot\\ F_n\end{bmatrix}$

gives the results :

$$F_i(x_1,x_2, \dots ,x_n) = \emptyset_{i,j} \quad \text{when} \quad x_k = a_{k,j} \quad , \begin{matrix}1\leqslant i\leqslant n \ ,\\ 1\leqslant k\leqslant n \ .\end{matrix}$$

For $i \neq h$: $F_i(x_1,x_2, \dots ,x_n) = a_{i,j}$ when $x_k = a_{k,j}$,

and since $a_{i,j} \triangleq x_i$, $1\leqslant j\leqslant 2^n$, then $F_i(x_1,x_2, \dots ,x_n) = x_i$.

For $i=h$: $F_h(x_1,x_2, \dots ,x_n) = \bar{a}_{h,j}$ when $x_i = a_{i,j}$,

and since $a_{i,j} \triangleq x_i$, $1\leqslant j\leqslant 2^n$, then $F_h(x_1,x_2, \dots ,x_n) = \bar{x}_h$.

If , in the equation $[C][\emptyset]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix}$,$[\emptyset]$ is of this form then the functions represented by the matrix $[C]$ will be the function represented by $[D]$ but re-defined upon the variables $(x_1,x_2, \dots ,\bar{x}_h, \dots ,x_n)$ instead of $(x_1,x_2, \dots ,x_h, \dots x_n)$.

Consider the effect of making $[\emptyset]$ identical to $[A]$ save that the h th row of $[\emptyset]$ is made equal to the g th row of $[A]$ and vice-versa.

Applying similar arguments to those used above it can be shown that , in the equation $[C][\emptyset]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\ x_n\end{bmatrix}$ the functions represented by the matrix $[C]$ will be exactly the functions represented by the matrix $[D]$ but re-defined upon the variables $(x_1, x_2, \ldots, x_h, x_g, \ldots x_n)$ instead of $(x_1, x_2, \ldots, x_g, x_h, \ldots, x_n)$.

In general it can be shown that if, in $[C][\emptyset]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = [D]\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$,

the rows of the post-multiplicative operator $[\emptyset]$ consist of a permutation of the rows of the $[A]$ matrix complemented or un-complemented, then the functions represented by $[C]$ will be those functions represented by $[D]$ but re-defined in terms of the same permutations and complementations of the defining variables corresponding to those rows.

A simple example of a post-multiplicative operator matrix constructed as the $[A]$ matrix but with certain rows complemented is :

$$[\emptyset] = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

, where the first and third rows are the complements of the first and third rows of the $[A]$ matrix and the second row is identical to the second row of the $[A]$ matrix.

Then $$[\emptyset]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \bar{x}_1 \\ x_2 \\ \bar{x}_3 \end{bmatrix}$$

If $[\emptyset]$ post-multiplies a single function matrix $[C]$, where $[C] = [0\ 1\ 1\ 0\ 1\ 0\ 1\ 0]$; writing $[\emptyset]$ in decimal notation gives :

$$\begin{array}{c} [0\ 1\ 1\ 0\ 1\ 0\ 1\ 0] \\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7} \end{array}[5\ 4\ 7\ 6\ 1\ 0\ 3\ 2]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [D]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

whence $[D] = [0\ 1\ 0\ 1\ 1\ 0\ 0\ 1]$.

To show that the function represented by $[C]$ is in fact a re-definition of the function represented by $[D]$, with x_1 replaced by $\bar{x}_1$ and x_3 replaced by $\bar{x}_3$, construct the Karnaugh map for $[D]$:

$x_3 \backslash x_1,x_2$	00	01	11	10
0	0	0	0	1
1	1	1	1	0

Now replace x_1 by $\bar{x}_1$ (this constitutes a reflection of the map about the axes which separate x_1 from $\bar{x}_1$) :

$x_3 \backslash x_1,x_2$	00	01	11	10
0	1	0	0	0
1	0	1	1	1

Finally replace x_3 by $\bar{x}_3$:

$x_3 \backslash x_1,x_2$	00	01	11	10
0	0	1	1	1
1	1	0	0	0

If this is re-expressed as a matrix $:\begin{bmatrix}0 & 1 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}$ it is seen to be equal to $\begin{bmatrix}C\end{bmatrix}$. See also Fig.8.

An example of a post-multiplicative operator consisting of a row permutation of the $\begin{bmatrix}A\end{bmatrix}$ matrix (without complementation) appears in Fig.9.

Of course $\begin{bmatrix}\emptyset\end{bmatrix}$ may be constructed of both permutations and complementations of the $\begin{bmatrix}A\end{bmatrix}$ matrix simultaneously . An example of this type of operator appears in Fig.10.

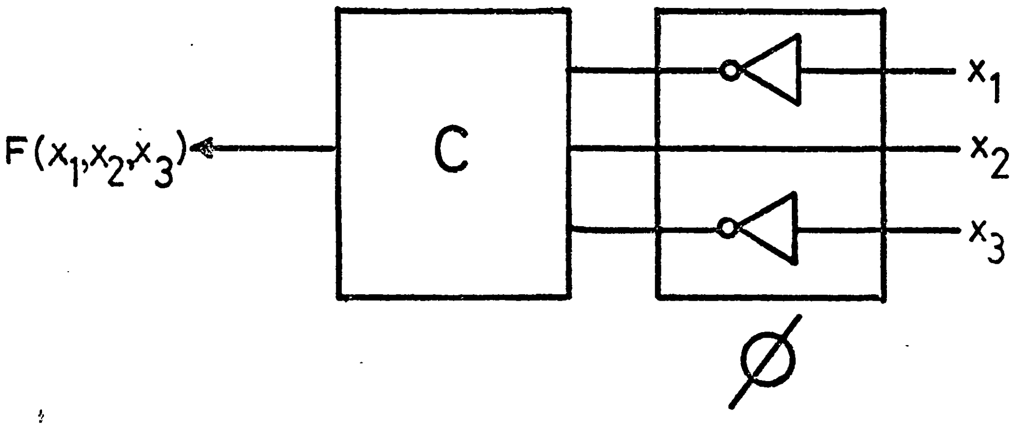
Not all non-singular post-multiplicative operator matrices can be categorised under variable complementation or interchange but these operators are the most useful, not only in terms of the representation of circuit synthesis but also Boolean function classification, see also Section 2.4 .

1.4.2 <u>Pre-multiplicative operators.</u>

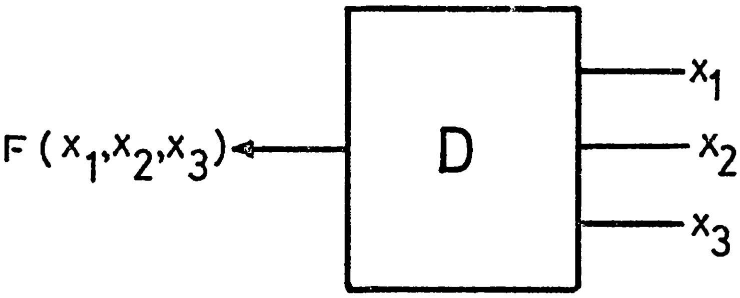
Consider the matrix equation

$$\begin{bmatrix}\emptyset\end{bmatrix}\begin{bmatrix}C\end{bmatrix}\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\x_n\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}\begin{bmatrix}x_1\\x_2\\ \cdot\\ \cdot\\x_n\end{bmatrix} \quad .$$

Now $\begin{bmatrix}\emptyset\end{bmatrix}$ pre-multiplies $\begin{bmatrix}C\end{bmatrix}$ and will be called a pre-multiplicative operator.



or

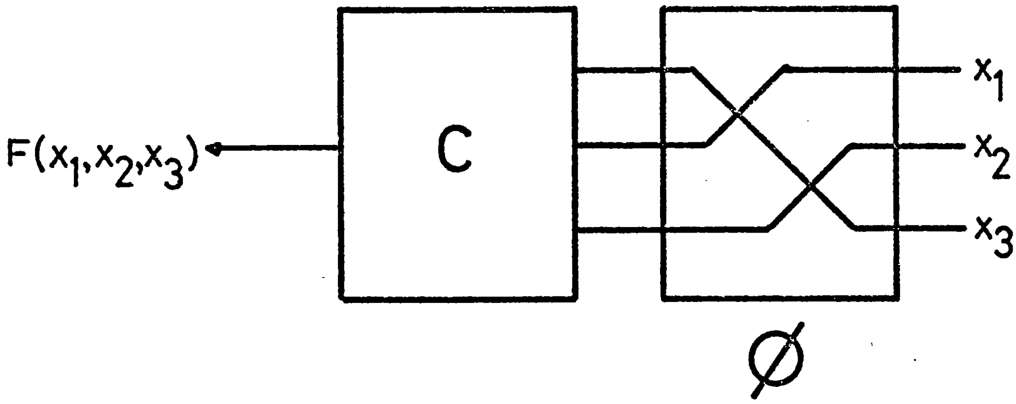


Implements $$[C][\emptyset]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$$

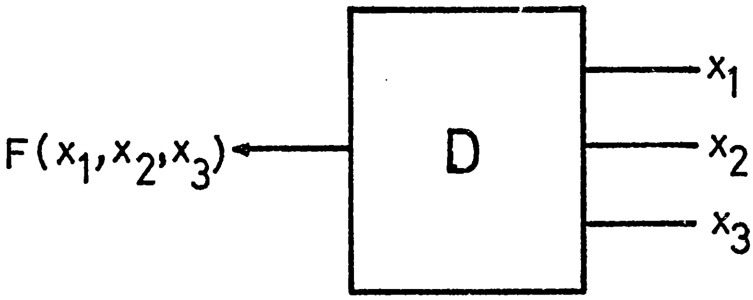
where

$$[\emptyset] = \begin{bmatrix}1&1&1&1&0&0&0&0\\0&0&1&1&0&0&1&1\\1&0&1&0&1&0&1&0\end{bmatrix}$$

Fig. 8



or

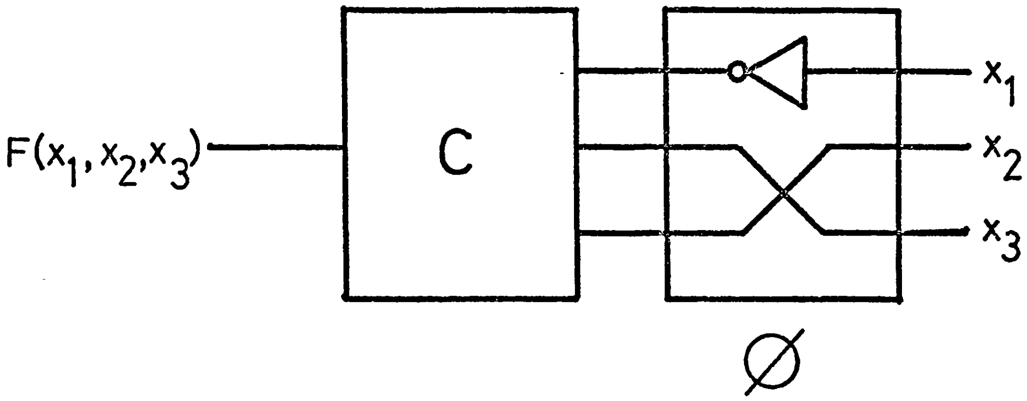


Implements $$[C][\varnothing]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$$

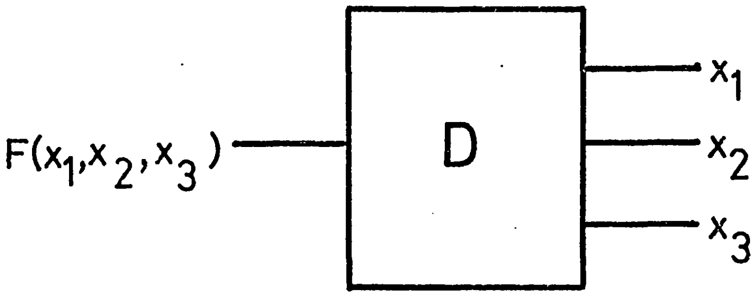
where

$$[\varnothing] = \begin{bmatrix}0&1&0&1&0&1&0&1\\0&0&0&0&1&1&1&1\\0&0&1&1&0&0&1&1\end{bmatrix}$$

Fig.9



or



Implements $$[C][\emptyset]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = [D]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$$

where

$$[\emptyset] = \begin{bmatrix}1&1&1&1&0&0&0&0\\0&1&0&1&0&1&0&1\\0&0&1&1&0&0&1&1\end{bmatrix}$$

Fig.10

From equation (1.14)

$$d_{r,j} = \phi_{r,m} \text{ when } a_{i,m} = c_{i,j} \text{ (over the allowed limits)}$$

Suppose that $[\phi] = [A]$ then in the above equation $[D] = [C]$.

Alternatively if $\phi_{r,m} = \bar{a}_{i,m}$, $1 \leqslant m \leqslant 2^n$, then $d_{r,j} = \bar{c}_{i,j}$, $1 \leqslant j \leqslant 2^n$, that is if the r th row of $[\phi]$ is equal to the complement of the i th row of $[A]$ then the r th row of $[D]$ will be equal to the complement of the i th row of $[C]$.

This approach can be extended to include $\phi_{r,m} = F(a_{i,m}, a_{k,m})$, $1 \leqslant m \leqslant 2^n$, where F is some logical function, then $d_{r,j} = F(c_{i,j}, c_{k,j})$. Then if the r th row of $[\phi]$ is some logical function of the i th and k th rows of $[A]$ then the r th row of $[D]$ will be the same logical function of those functions defined by the i th and k th rows of $[C]$.

These observations show that the pre-multiplicative operators allow the manipulation of whole logical functions.

For example consider the equation :

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

, and suppose that three other functions are required , namely $F_4 = F_1 \cap F_2$, $F_5 = \bar{F}_3 \cup F_1$ and $F_6 = F_1$. These functions may be evaluated as follows :

Using the general equation
$$[\phi][C] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = [D] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} ;$$

then
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} F_4 \\ F_5 \\ F_6 \end{bmatrix}$$

from the $[A]$ matrix
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} .$$

This evaluation is derived in the following way :

The first row of the pre-multiplying matrix $[\phi]$ is equal to the intersection of the first and second rows of $[A]$. The second row of $[\emptyset]$ is equal to the union of the complement of the third row of $[A]$ with the first row of $[A]$. The third row of $[\emptyset]$ is equal to the first row of $[A]$. The implementation of this example is given in Fig.11.

The consideration of pre- and post-multiplicative operators together with their associated properties is essential in the interpretation of both the advantages and versatility of Boolean matrix algebra. They will be refered to again when circuit synthesis is considered in later sections.

1.4.3 Operations of the Parallel Composition Type.

Another useful class of operators are those of the parallel composition type. These are written as

$$[B] \$ [C] \qquad \ldots (1.35)$$

where $\$$ signifies the logical manipulation of the matrices $[B]$ and $[C]$ on an element by element basis.

For example $\left[[B] \cup [C] \right] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = [D] \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$ signifies

$d_{i,j} = b_{i,j} + c_{i,j}$ over the dimensional limits.

$[B][C]$ and $[D]$ have the same dimensions.

This example may be interpreted as shown in Fig. 12.

It is also possible to apply different operators to different rows of the matrices which are to undergo parallel composition, giving rise to equations such as

$$\left[[B] \overset{\cup}{\cap} [C] \right] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [D] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{where}$$

$d_{1,j} = b_{1,j} + c_{1,j}$ and $d_{2,j} = b_{2,j} \cdot c_{2,j}$

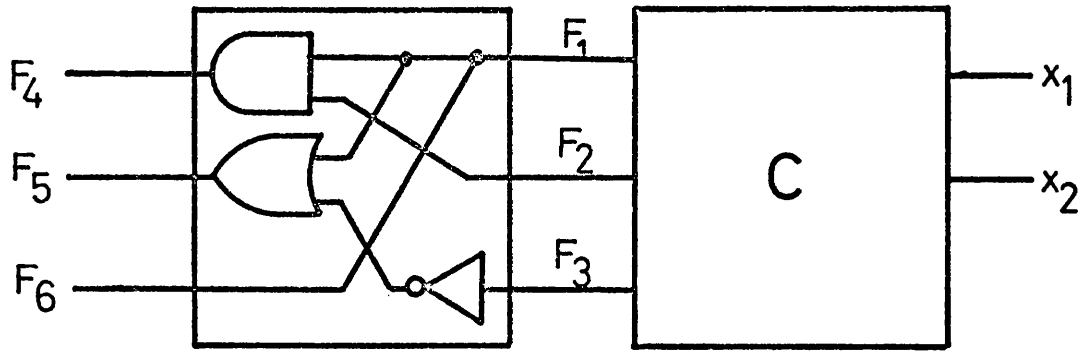


Fig. 11

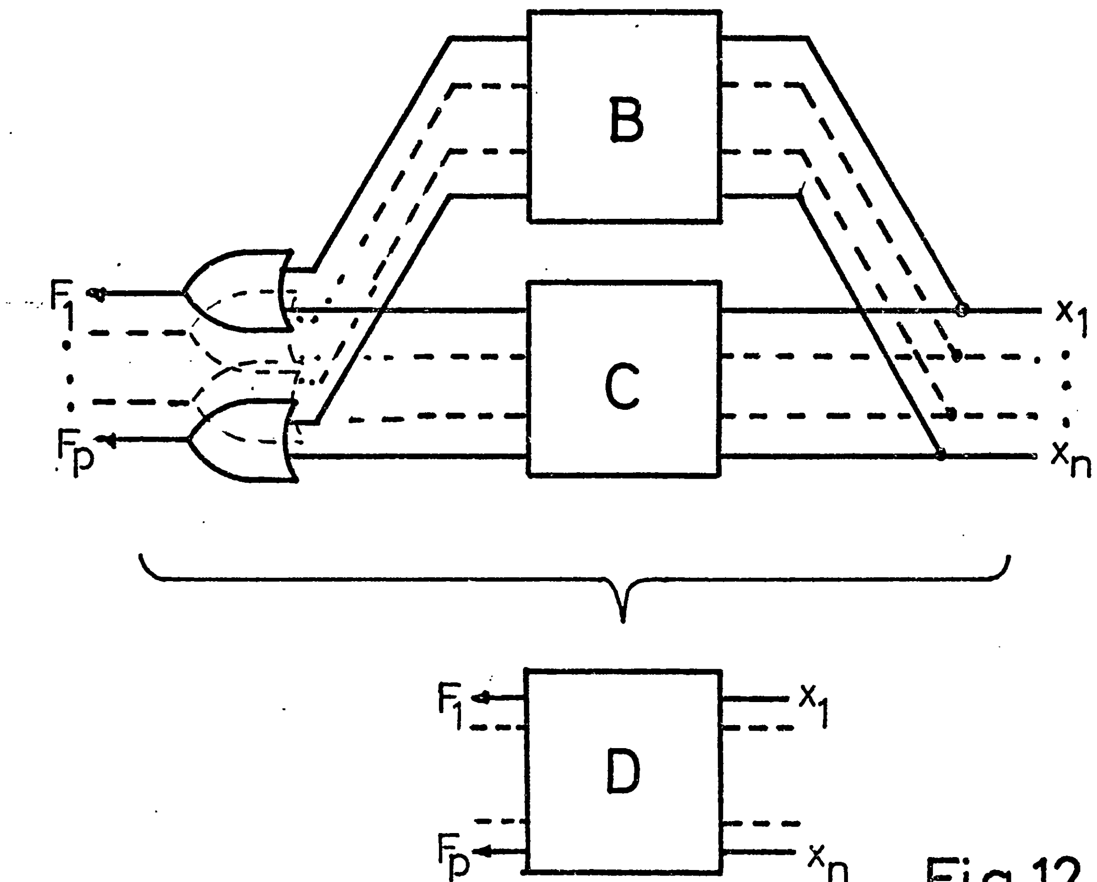


Fig.12

which results in the circuit of Fig. 13 .

This type of operator is used in the extraction of prime implicants , see Section 1.5.6.

1.5 Practical Applications.

1.5.1 Introduction.

It has already been demonstrated that Boolean matrices provide an excellent method of evaluating both the logical transfer functions and topology of multi-output combinational logic circuits. It is now possible , by means of worked examples whenever possible , to show the special importance of certain of the properties of Boolean matrix algebra developed above , in the analysis and synthesis of logic circuits.

1.5.2 Matrix Multiplication.

Worked example.

Given Two logic modules $[B]$ and $[C]$ have been designed according to the specifications :

$$[B] = \begin{bmatrix} 1 & * & 0 & 1 & * & 1 & 0 & 0 \\ 0 & * & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & * & 0 \end{bmatrix}$$

$$[C] = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & * & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & * & 0 & 1 \end{bmatrix}$$

A system specification is given by $[D]$ where

$$[D] = \begin{bmatrix} 1 & 1 & 1 & * & 0 & 0 & 0 & 1 \\ 1 & * & 0 & 0 & 1 & 0 & * & 0 \\ 1 & 1 & 1 & * & 1 & * & * & 0 \end{bmatrix}$$ (* signifies don't care)

Is it possible to synthesise this system by cascading the modules represented by $[B]$ and $[C]$?

Solution

Try cascading $[B]$ and $[C]$ as $\left[B \,\vdots\, C\right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [E] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$.

In decimal notation this is

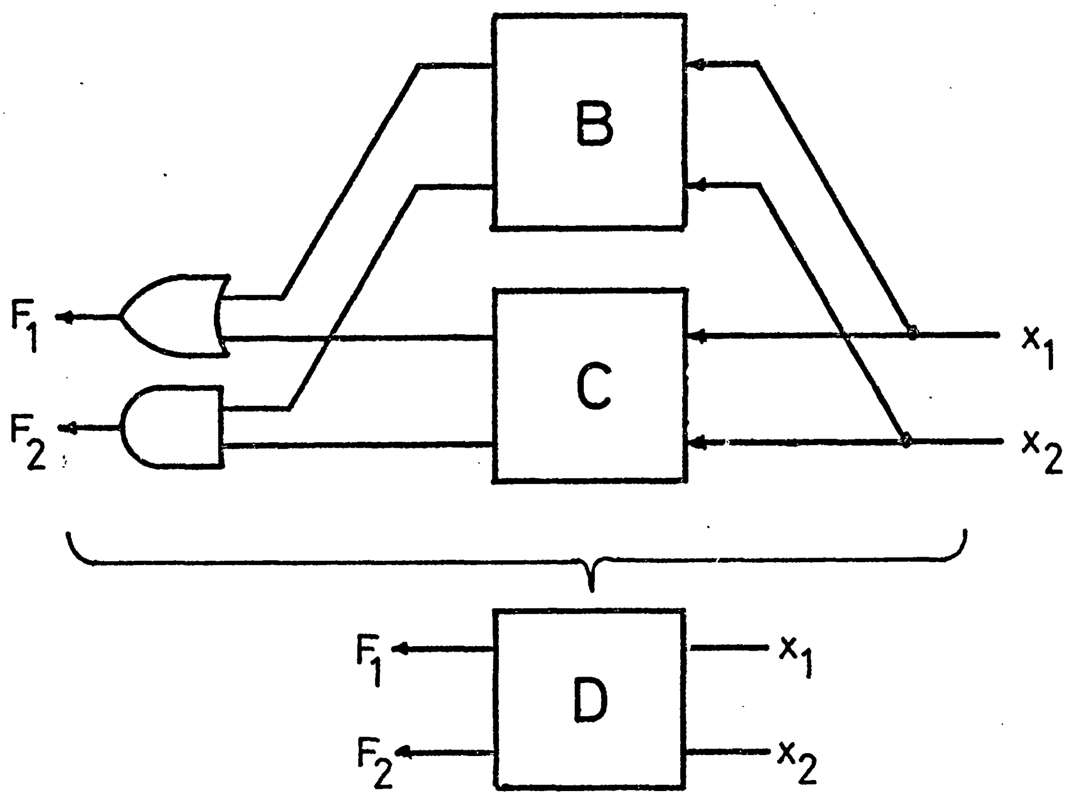


Fig. 13

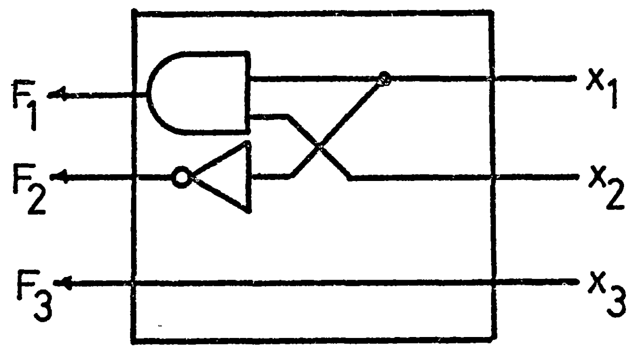


Fig. 14

$$\left[5\ \begin{array}{c}1\\\hline 3\\\hline 5\\\hline 7\end{array}\ 3\ 7\ \frac{0}{4}\ 4\ \frac{0}{1}\ 0\right]\left[3\ 0\ 0\ \frac{4}{6}\ 2\ \frac{6}{7}\ 6\ 5\right]\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right] = \left[7\ 5\ 5\ \begin{array}{c}0\\\hline 1\\\hline 4\end{array}\ 3\ \frac{0}{1}\ \frac{0}{1}\ 4\right]\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right]$$

0 1 2 3 4 5 6 7

Comparing $[E]$ with $[D]$ (in decimal notation) :

$$[D] = \left[7\ \frac{5}{7}\ 5\ \begin{array}{c}0\\\hline 1\\\hline 4\\\hline 5\end{array}\ 3\ \frac{0}{1}\ \begin{array}{c}0\\\hline 1\\\hline 2\\\hline 3\end{array}\ 4\right]$$

$$[E] = \left[7\ 5\ 5\ \begin{array}{c}0\\\hline 1\\\hline 4\end{array}\ 3\ \frac{0}{1}\ \frac{0}{1}\ 4\right]$$

it can be seen that $[E]$ falls within the specification of $[D]$.

It may therefore be concluded that the proposed method of cascading the modules will indeed synthesise the system.

This example illustrates how Boolean matrices may be used to advantage in the synthesis of partially specified systems.

1.5.3 Inverse Matrices.

Worked example 1 .

A logic system has been designed . It has been decided to extend the capabilities of the system by producing three extra outputs specified by $[D]$, where

$$[D]\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right] = \begin{bmatrix}1&0&1&1&1&1&1&1\\ *&0&1&1&0&0&0&1\\ 1&1&1&1&0&*&1&1\end{bmatrix}\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right] .$$

It has been suggested that these three outputs may be generated from two outputs already available and specified by the matrix C , where

$$[C]\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right] = \begin{bmatrix}0&1&0&0&1&*&0&0\\ *&0&1&1&1&0&0&1\end{bmatrix}\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right]$$

Is this possible ? - If so find the required module.

Solution

Represent the problem as

$$[B][C]\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right] = [D]\left.\begin{matrix}x_1\\x_2\\x_3\end{matrix}\right]$$

Substitute the given information (decimal notation) :

$$\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} \substack{0\\1} & 2 & 1 & 1 & 3 & \substack{0\\2} & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \substack{5\\7} & 1 & 7 & 7 & 4 & \substack{1\\5} & 5 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Check the validity of the equation :

1/ The equation is dimensionally correct.

2/ Criterion 1 shows that the equation is conditionally valid where

$$\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} \substack{0\\1} & 2 & 1 & 1 & 3 & \substack{0\\2} & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \substack{5\\7} & 1 & 7 & 7 & 4 & \substack{5\\1} & 5 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The outputs may therefore be generated.

Now the matrix $\begin{bmatrix} B \end{bmatrix}$ can be evaluated by inspection by noting that in $\begin{bmatrix} B \end{bmatrix}$: the value at n-tuple 0 must have the value '5'

1 must have the value '7'

2 must have the value '1'

3 must have the value '4'

viz.

$$\begin{array}{cccc} \begin{bmatrix} 5 & 7 & 1 & 4 \end{bmatrix} \\ \mathbf{0}\ \mathbf{1}\ \mathbf{2}\ \mathbf{3} \end{array}\begin{bmatrix} \substack{0\\1} & 2 & 1 & 1 & 3 & \substack{0\\2} & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \substack{5\\7} & 1 & 7 & 7 & 4 & \substack{5\\1} & 5 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Alternatively the singular inverse may be calculated for $\begin{bmatrix} C \end{bmatrix}$, from $\begin{bmatrix} C \end{bmatrix}\begin{bmatrix} C \end{bmatrix}^{-1} = \begin{bmatrix} A \end{bmatrix}$, where $\begin{bmatrix} C \end{bmatrix}$ may take any convenient allowed form.

ie.

$$\begin{array}{c} \begin{bmatrix} 0 & 2 & 1 & 1 & 3 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{0}\ \ \mathbf{1}\ \ \mathbf{2}\ \ \mathbf{3}\ \ \mathbf{4}\ \ \mathbf{5}\ \ \mathbf{6}\ \ \mathbf{7} \end{array}\begin{bmatrix} \substack{0\\5\\6} & \substack{2\\3\\7} & 1 & 4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix}$$

Then $\begin{bmatrix} B \end{bmatrix}$ may be calculated from $\begin{bmatrix} B \end{bmatrix} = \begin{bmatrix} D \end{bmatrix}\begin{bmatrix} C \end{bmatrix}^{-1}$ for the corresponding value of $\begin{bmatrix} D \end{bmatrix}$.

ie.

$$\begin{bmatrix} B \end{bmatrix} = \begin{array}{c} \begin{bmatrix} 5 & 1 & 7 & 7 & 4 & 5 & 5 & 7 \end{bmatrix} \\ \mathbf{0}\ \ \mathbf{1}\ \ \mathbf{2}\ \ \mathbf{3}\ \ \mathbf{4}\ \ \mathbf{5}\ \ \mathbf{6}\ \ \mathbf{7} \end{array}\begin{bmatrix} \substack{0\\5\\6} & \substack{2\\3\\7} & 1 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 7 & 1 & 4 \end{bmatrix}.$$

By either method the required module has been evaluated as :

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

<u>Worked example 2.</u>

A system specification is given by $\begin{bmatrix} D \end{bmatrix}$ where

$$\begin{bmatrix} D \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} .$$

It is proposed to employ an output module $\begin{bmatrix} B \end{bmatrix}$ to synthesise this system where

$$\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} .$$

Design a logic module to be placed before $\begin{bmatrix} B \end{bmatrix}$ which will synthesise the system . Can each output function of this module be synthesised separately ?

<u>Solution.</u>

Let the problem be represented as :

$$\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} C \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} D \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} .$$

Substitution of the given information (in decimal notation) gives :

$$\begin{bmatrix} 3 & 6 & 1 & 5 & 6 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} C \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 1 & 5 & 3 & 6 & 6 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Check the validity of the equation :

1/ The equation is dimensionally correct.

2/ Criterion 2 is satisfied .

Construct $\begin{bmatrix} B \end{bmatrix}^{-1}$ from $\begin{bmatrix} B \end{bmatrix}\begin{bmatrix} B \end{bmatrix}^{-1} = \begin{bmatrix} A \end{bmatrix}$:

$$\begin{array}{c} \begin{bmatrix} 3 & 6 & 1 & 5 & 6 & 2 & 0 & 0 \end{bmatrix} \\ \mathbf{0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7} \end{array}\begin{bmatrix} \frac{6}{7} & 2 & 5 & 0 & * & 3 & \frac{1}{4} & * \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}$$

Compute $\begin{bmatrix} C \end{bmatrix}$ from $\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix} B \end{bmatrix}^{-1}\begin{bmatrix} D \end{bmatrix}$:

$$\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix} \frac{6}{7} & 2 & 5 & 0 & * & 3 & \frac{1}{4} & * \end{bmatrix}\begin{bmatrix} 1 & 5 & 1 & 5 & 3 & 6 & 6 & 2 \end{bmatrix}$$
$$\mathbf{0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7}$$
$$= \begin{bmatrix} 2 & 3 & 2 & 3 & 0 & \frac{1}{4} & \frac{1}{4} & 5 \end{bmatrix}$$
$$\mathbf{0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7}$$

Now the value of $\left[C\right]$ at n-tuples 5 and 6 is $\substack{'1'\\ 4}$, or in vector form : $\begin{bmatrix}1\\0\\0\end{bmatrix}$ or $\begin{bmatrix}0\\0\\1\end{bmatrix}$. It follows that if the upper function has the value '0' at these n-tuples then the lower function must have the value '1' and vice-versa. The functions are therefore dependent and cannot be synthesised separately.

If $\left[C\right]$ is chosen to have the form

$$\begin{bmatrix}2 & 3 & 2 & 3 & 0 & 1 & 4 & 5\end{bmatrix} \equiv \begin{bmatrix}0&0&0&0&0&0&1&1\\1&1&1&1&0&0&0&0\\0&1&0&1&0&1&0&1\end{bmatrix}$$

then $\left[C\right]\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = \begin{bmatrix}F_1\\F_2\\F_3\end{bmatrix}$ may be synthesised by constructing the Karnaugh maps :

x_3 \ x_1,x_2	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$F_1(x_1,x_2,x_3)$
$= x_1 \cdot x_2$

x_3 \ x_1,x_2	00	01	11	10
0	1	1	0	0
1	1	1	0	0

$F_2(x_1,x_2,x_3)$
$= \bar{x}_1$

x_3 \ x_1,x_2	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$F_3(x_1,x_2,x_3)$
$= x_3$

The corresponding circuit implementation is shown in Fig.14.

1.5.4 Matrices Raised to Exponents.

Worked example.

Tests have been carried out on a cascade of logic modules , each defined by the matrix $\left[C\right]$. The overall transfer function of five such cascaded modules gives the result :

$$\begin{bmatrix}5 & 1 & 2 & 3 & 4 & 0 & 6 & 7\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$$

and the overall transfer function of three such modules gives the result :

$$\begin{bmatrix}5 & 2 & 3 & 4 & 7 & 0 & 6 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} .$$

Find the matrix which defines the transfer function of one such module .

Solution.

Now $[C]^5$ and $[C]^3$ are known. First compute $[C]^2$ from $[C]^2 = [C]^5\,[C]^{-3}$.

Find $[C]^{-3}$ from $[C]^3\,[C]^{-3} = [A]$:

$$[5\;2\;3\;4\;7\;0\;6\;1][5\;7\;1\;2\;3\;0\;6\;4] = [0\;1\;2\;3\;4\;5\;6\;7]$$

ie. $[C]^{-3} = [5\;7\;1\;2\;3\;0\;6\;4]$

Then $[C]^2 = [C]^5\,[C]^{-3} = [5\;1\;2\;3\;4\;0\;6\;7][5\;7\;1\;2\;3\;0\;6\;4]$

$$= [0\;7\;1\;2\;3\;5\;6\;4]$$

Second compute $[C]$ from $[C] = [C]^3\,[C]^{-2}$.

Find $[C]^{-2}$ from $[C]^2\,[C]^{-2} = [A]$:

$$\begin{array}{c}[0\;7\;1\;2\;3\;5\;6\;4]\\ \mathbf{0\;1\;2\;3\;4\;5\;6\;7}\end{array}[0\;2\;3\;4\;7\;5\;6\;1] = [0\;1\;2\;3\;4\;5\;6\;7]$$

ie. $[C]^{-2} = [0\;2\;3\;4\;7\;5\;6\;1]$

Then $[C] = [C]^3\,[C]^{-2} = \begin{array}{c}[5\;2\;3\;4\;7\;0\;6\;1]\\ \mathbf{0\;1\;2\;3\;4\;5\;6\;7}\end{array}[0\;2\;3\;4\;7\;5\;6\;1]$

$$= [5\;3\;4\;7\;1\;0\;6\;2].$$

Which is the required result.

The known power table is then.

MATRIX	π
.	.
.	.
5 7 1 2 3 0 6 4	-3
0 2 3 4 7 5 6 1	-2
?	-1
0 1 2 3 4 5 6 7	0
5 3 4 7 1 0 6 2	1
0 7 1 2 3 5 6 4	2
5 2 3 4 7 0 6 1	3
?	4
5 1 2 3 4 0 6 7	5
.	.
.	.
.	.

1.5.5 Representation of Iterative Cascades.

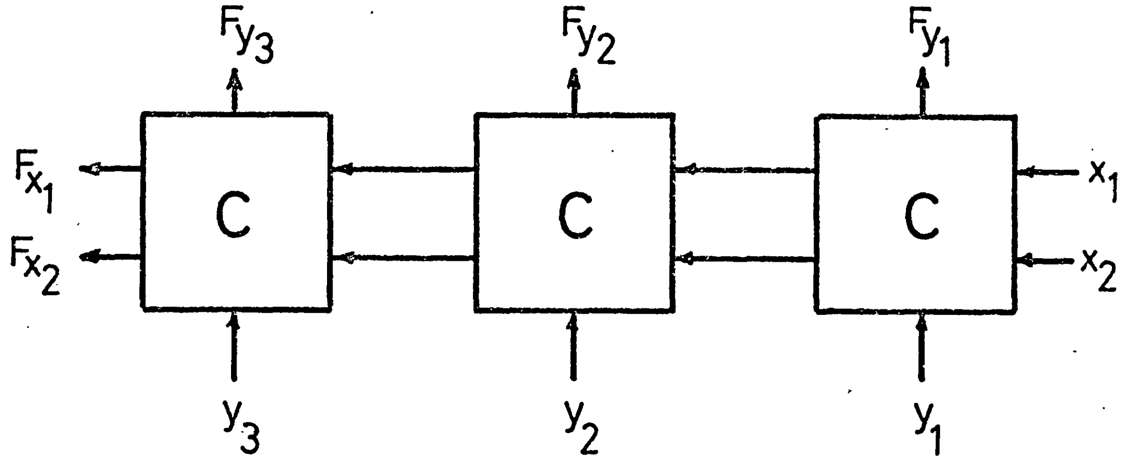
The concept of an iterative cascade of logic modules of the type shown in Fig. 6 . has already been introduced together with the associated matrix representation. See p 39.

The representation of cascades of the type shown in Fig.7. p 40 , will now be considered.

A simplified version of this cascade appears in Fig.15. In order to distinguish easily the direction of signal flows in this cascade the horizontal flows, input/output , are labelled x_1, x_2 / $F_{x_1}, F_{x_2}, F_{x_3}$ and the vertical flows , input/output , are labelled y_1, y_2, y_3 / $F_{y_1}, F_{y_2}, F_{y_3}$ respectively.

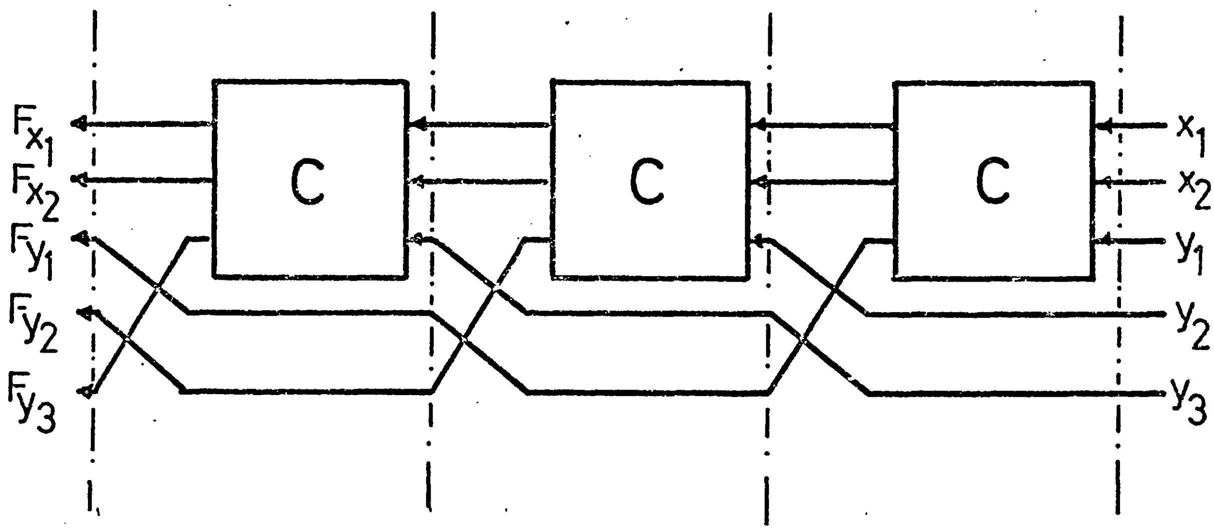
Such arrays have been considered by Hennie , see reference 6 , and can be shown to be transformable to ideal finite - state machines. The inputs x_1, x_2 are termed the starting state of the cascade and the corresponding inputs to the second logic module are called the next , or second , state and so on. In general the starting state of the cascade is fixed for a particular application and the cascade is used to compute a function of the input variables y_1, y_2, y_3 . It is not the purpose here however to investigate the general properties of such cascades , but to show that they may be expressed in matrix form. Reference 7 should be consulted for a detailed treatment of the properties of cascaded iterative arrays.

Now the type of iterative cascade which has been shown to be easily represented by Boolean matrices heretofore is that of Fig.6. Comparison between the cascade of Fig.6 and that of Fig. 7 shows that they differ in that the former case has a single (horizontal) flow path wheras the latter has two flow paths (horizontal and vertical). At first sight it would appear that the cascade of Fig.7 is not amenable to Boolean matrix representation because each module of the cascade is furnished with a unique input y_1, y_2, y_3 etc.
To show that this problem is surmountable consider the circuit of Fig.16 which is an alternative representation of the simple cascade of Fig.15. The inputs/outputs : y_1, y_2, y_3 / $F_{y_1}, F_{y_2}, F_{y_3}$ have been re-orientated so that they are applied in a horizontal direction,



Fy3
Fy2
Fy1
Fx1
Fx2
C
C
C
x1
x2
y3
y2
y1

Fig.15



Fx1
Fx2
Fy1
Fy2
Fy3
C
C
C
x1
x2
y1
y2
y3

Fig. 16

and between each module is a simple crossover network which enables each input to be applied to the correct module and also enables the corresponding output to appear correctly orientated at the termination of the cascade ; moreover each crossover network is identical. Now it has already been shown that a crossover network of the type shown in Fig.16 may be represented by a pre-multiplicative matrix operator.

The cascade of Fig.16 may therefore be represented by the equation :

$$[\phi][c][\phi][c][\phi][c]\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} F_{x_1} \\ F_{x_2} \\ F_{y_1} \\ F_{y_2} \\ F_{y_3} \end{bmatrix} \quad \text{, or}$$

$$\left[[\phi][c]\right]^3 \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} F_{x_1} \\ F_{x_2} \\ F_{y_1} \\ F_{y_2} \\ F_{y_3} \end{bmatrix} .$$

The operator $[\phi]$ will have the form :

$$\begin{aligned} &\phi_{1,j} = a_{1,j} , \quad \phi_{2,j} = a_{2,j} , \\ &\phi_{3,j} = a_{4,j} , \quad \phi_{4,j} = a_{5,j} , \qquad 1 \leqslant j \leqslant 2^5 \\ &\phi_{5,j} = a_{3,j} . \end{aligned}$$

This technique can be applied to any cascade of the type shown in Fig.7. including such cascades having multiple y inputs/outputs for each module.

The finite-state machine corresponding to the cascade of Fig.15 is shown in Fig.17. The horizontal , or x inputs , to the combinational logic module being initially applied to give the starting state , and each y_i , i=1,i=2,i=3 being applied to the module at times T=1 , T=2 and T=3 respectively.

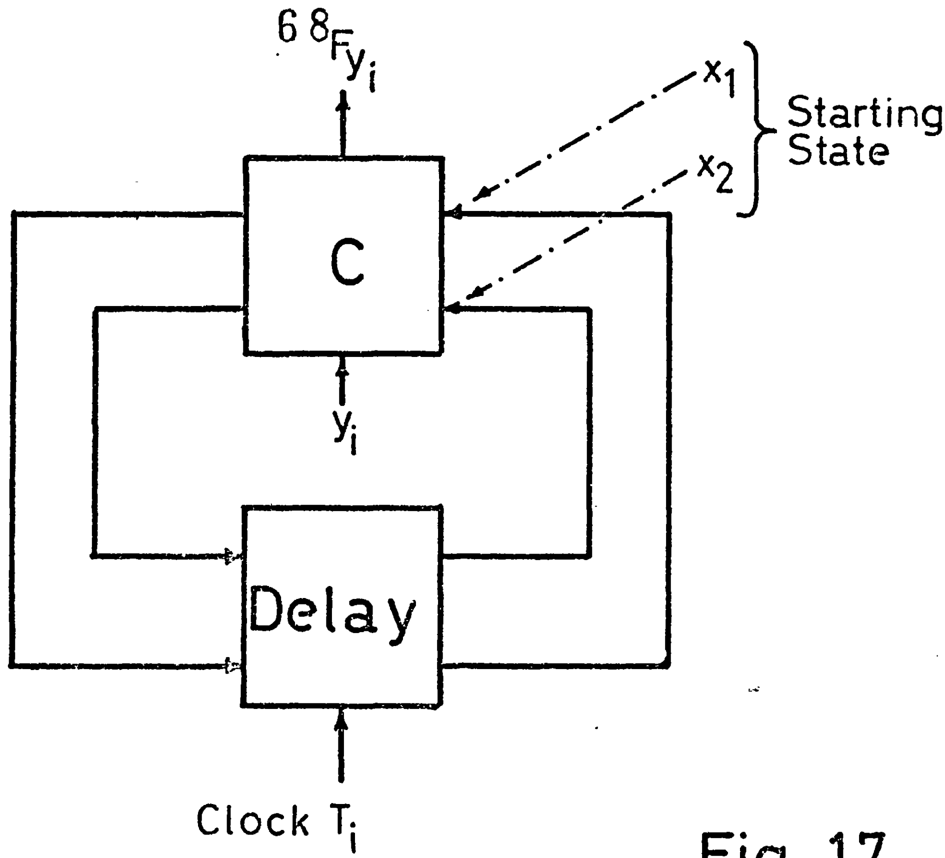


Fig. 17

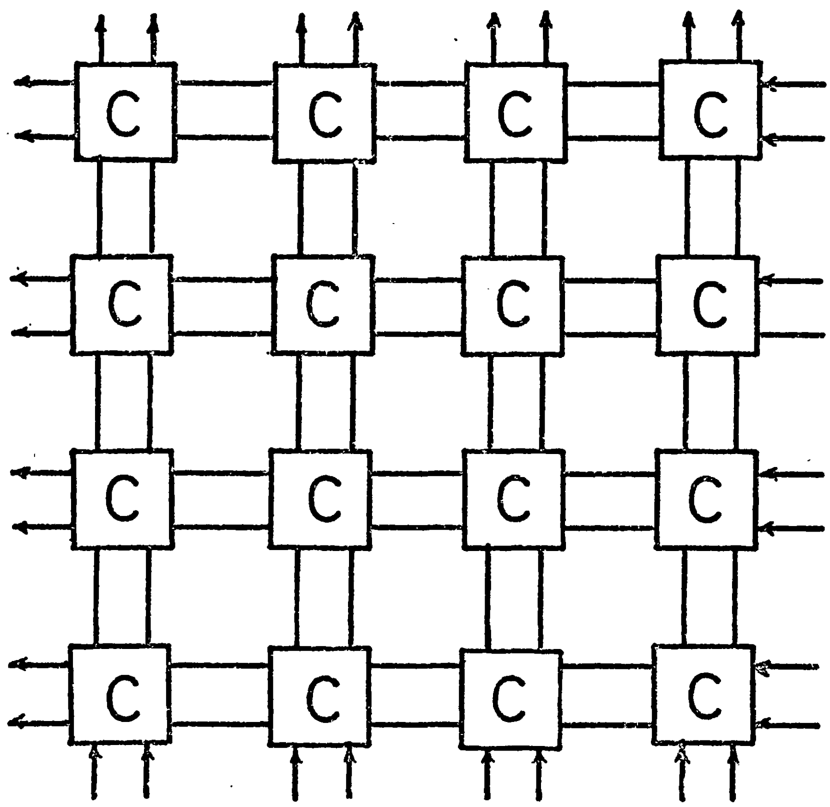


Fig.18

This example illustrates that Boolean matrices may be used to represent finite-state machines.†

It is also possible to show that Boolean matrices may be used to represent arrays of the type shown in Fig.18.

This method of representing iterative cascades has the disadvantage that it is limited by the large size of matrices necessary to represent long cascades or large arrays.

Because of the difficulties described in extracting the roots of Boolean matrices they are not readily applicable to the synthesis of such systems.

1.5.6 Extraction of the Prime Implicants of Functions.

Consider a Boolean function $F(x_1,x_2, \ldots ,x_i, \ldots ,x_n)$, let this function be denoted by F(X).

Take the function derived from the function above by complementing the variable x_i ; let this function be denoted by $F_{\bar{i}}(X)$.

$$\text{Let} \quad F_i(X) \triangleq F(X) \cap F_{\bar{i}}(X) \qquad \ldots (1.36)$$

Now $F_i(X)$ constitutes the true minterms of F(X) which are independent of the variable x_i ; that is $F_i(X)$ may be defined upon the variables $(x_1,x_2, \ldots ,-, \ldots ,x_n)$ alone. If F(X) has no true minterms independent of x_i then $F_i(x) = \theta$, where θ is a null set.

Now for any function F(X) , each $F_i(X)$, $1 \leqslant i \leqslant n$, $F_i(x) \neq \theta$, will contain true minterms which lie in pairs of adjacent states. If these minterms are plotted on a Karnaugh map they will fall into squares with adjacent sides. This must be so since such minterms differ only in the complementation of one defining variable.

Each $F_i(X)$ will be called a partition of F(X).

It is therefore possible to generate n such partitions, each partition containing terms independent of a partcular variable.

---

† With certain restrictions

Clearly , if the function contains adjacent terms which are independent of both x_i and x_j :

$$F_i(X) \cap F_j(X) \neq \theta \quad , \; i \neq j \; , \quad 1 \leqslant i,j \leqslant n \quad ..(1.37)$$

In itself the partition listing outlined above establishes whether a function is 'reducible' (it partitions in at least one variable) and if so in which variables this is possible. If a function fails to partition in every one of its defining variables it is irreducible.

As will be shown shortly , the manipulation of such a set of partitions enables the function to be reduced to a number of prime -implicants.

Now the partitions $F_i(X)$, $1 \leqslant i \leqslant n$, may be generated using the post-multiplicative Boolean matrix operators which have previously been developed. Moreover each partition may be evaluated for several functions simultaneously. The functions to be partitioned are defined by $\left[C\right]$ where

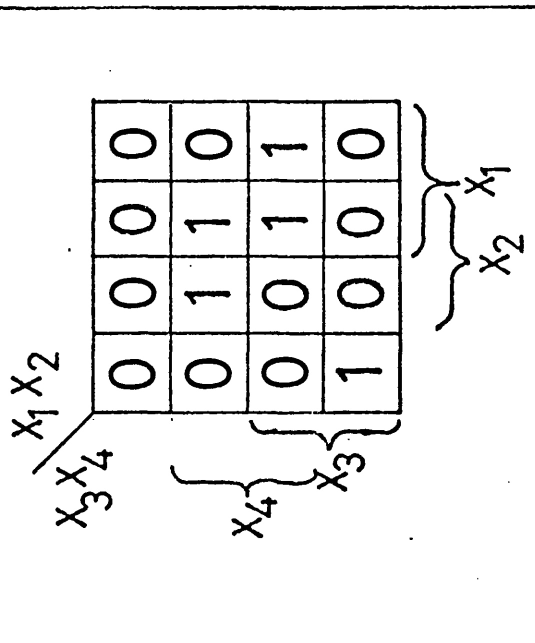
$$\left[C\right]\left[\emptyset\right]\left.\begin{matrix} x_1 \\ \cdot \\ \cdot \\ x_i \\ \cdot \\ \cdot \\ x_n \end{matrix}\right] = \left[D\right]\left.\begin{matrix} x_1 \\ \cdot \\ \cdot \\ x_i \\ \cdot \\ \cdot \\ x_n \end{matrix}\right]$$

, and $\left[\emptyset\right]$ is an operator matrix identical to the $\left[A\right]$ matrix except that the row $a_{i,j}$, $1 \leqslant j \leqslant 2^n$, is complemented. $\left[D\right]$ then defines $F_{\bar{i}}(X)$ for each of the functions specified by $\left[C\right]$.

If the parallel composition $\left[C\right] \cap \left[D\right]$ is evaluated then the result will be equal to $F_i(X) = F_{\bar{i}}(X) \cap F(X)$ for each function defined by $\left[C\right]$.

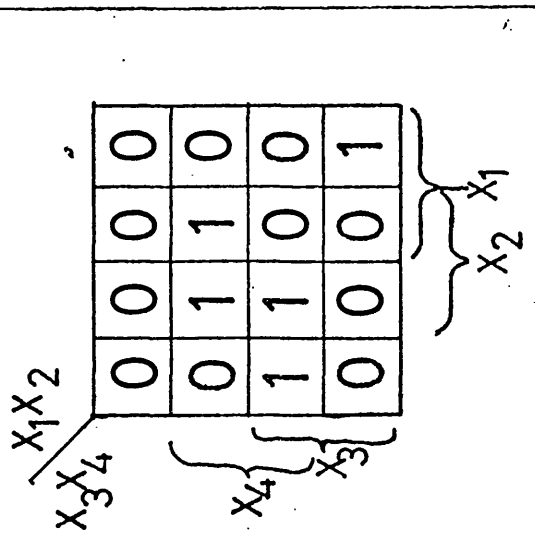
Example.

As an example of the extraction of the partitions $F_i(X)$, $1 \leqslant i \leqslant n$, applied to a single function , consider the function shown in Fig. 19a.



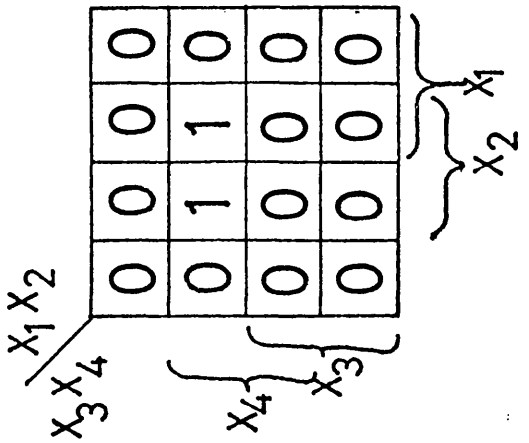
$F(X)$

FIG19a



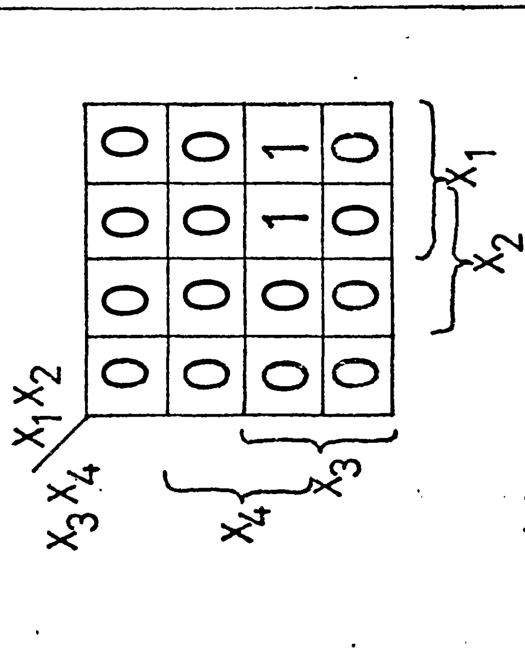
$F_{\bar{1}}(X)$

FIG19b



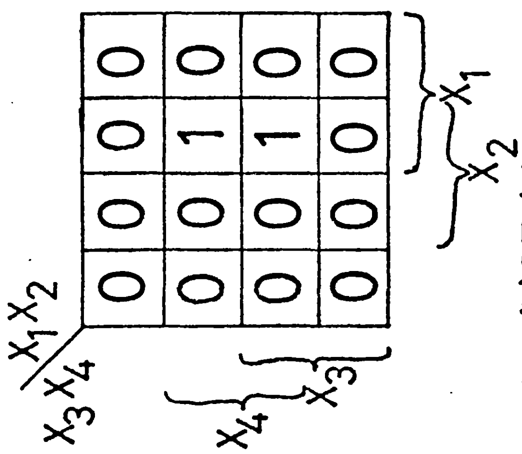
$F_1(X)=F(X)\cap F_{\bar{1}}(X)$

FIG19c



$F_2(X)=F(X)\cap F_{\bar{2}}(X)$

FIG.19d



$F_3(X)=F(X)\cap F_{\bar{3}}(X)$

FIG 19e.

$F_{\bar{1}}(X)$ is computed from $\begin{bmatrix}C\end{bmatrix}\begin{bmatrix}\emptyset\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}$, where

$\begin{bmatrix}C\end{bmatrix}$ defines the given function and $\begin{bmatrix}\emptyset\end{bmatrix}$ is identical to $\begin{bmatrix}A\end{bmatrix}$ save that the first row is complemented. $\begin{bmatrix}D\end{bmatrix}$ then specifies $F_{\bar{1}}(X)$.

viz.

$$\begin{bmatrix}0010010000010101\end{bmatrix}\begin{bmatrix}1111111100000000\\0000111100001111\\0011001100110011\\0101010101010101\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}$$

or in decimal notation

$$\begin{array}{c}\begin{bmatrix}0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\end{bmatrix}\\0\,1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\end{array}\begin{bmatrix}8\,9\,10\,11\,12\,13\,14\,15\,0\,1\,2\,3\,4\,5\,6\,7\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}$$

whence $\begin{bmatrix}D\end{bmatrix} = \begin{bmatrix}0001010100100100\end{bmatrix}$.

The corresponding function $F_{\bar{1}}(X)$ appears in Fig. 19b. It should be noted that this operation corresponds to a reflection of the function, as depicted by a Karnaugh map , about the axes which separate x_1 from $\bar{x}_1$.

Computing $F_{\bar{1}}(X) \cap F(X)$ from $\begin{bmatrix}D\end{bmatrix} \cap \begin{bmatrix}C\end{bmatrix}$ gives $\begin{bmatrix}0000010000000100\end{bmatrix}$ which is equal to $F_1(X)$, the result is shown in Fig. 19c.

If this procedure is repeated for the evaluation of $F_2(X)$ and $F_3(X)$ the results are as shown in Fig. 19d and Fig. 19e respectively. $F_4(X)$ can be shown to be a null set.

From these results it is clear that F(X) has a pair of true minterms independent of x_1 , a pair independent of x_2 and a pair independent of x_3 . In addition , one minterm is irreducible as it appears in none of the partitions.

F(X) may thus be expressed as :

$$F(X) = x_2.\bar{x}_3.x_4 + x_1.x_3.x_4 + x_1.x_2.x_4 + \bar{x}_1.\bar{x}_2.x_3.\bar{x}_4$$

Consider now the effect of re-partitioning $F_i(x)$ in terms of another variable x_j. Then if $F_{i,j}(X)$ defines this operation :

$$F_{i,j}(X) = F_i(X) \cap F_{\bar{j}}\left\{ F_i(X) \right\} \quad \ldots (1.38)$$

If this second partition exists, all terms it contains must lie adjacent in at least two variables. (Blocks of at least four true minterms adjacent when plotted on a Karnaugh map)

Suppose that all the possible one-variable partitions (P_1) of a function defined upon n variables are taken. Then $P_1 \triangleq \bigcup_{i=1}^{n} F_i(X)$ constitutes all true minterms of a function which are adjacent in at least one variable.

Then $F(X) \cap \bar{P}_1$ constitutes all terms having no adjacencies. (They are irreducible)

Suppose now that all possible two-variable partitions (P_2) of the function are taken. Then $P_2 = \bigcup_{0 \leqslant i < j \leqslant n} F_{i,j}$ constitutes all terms of the function which are adjacent in at least two variables.

Then $P_1 \cap \bar{P}_2$ constitutes all terms adjacent in one variable only. (They exist in pairs on a Karnaugh map)

This idea may be extended to $P_3, P_4, \ldots, P_n$.

It should be noted that the result of each $P_i \cap \bar{P}_{(i+1)}$ may be decoded into specific pairs, duo-pairs etc. by means of the partition variables leading to the result. Alternatively, specific decoding algorithms may be used.

The result of these operations is the extraction of the redundent and irredundent prime implicants of the function, and represents an attractive alternative method to that of Quine-McCluskey, see references 9,10.

In addition the function may be selectively analysed for its dependence upon any particular variable(s).

Fig. 20 shows the exhaustive partitioning map of a four-variable problem. The evaluation of all the partitions shown is sufficient to enable the evaluation of the redundent and irredundent prime implicants of any fourth order function. Note that once a map is generated in the form shown, removal of branches associated with γ of the variables reduces the map to that of order $(n-\gamma)$ without recourse to re-arrangement.

The number of partitions required for the solution of an n-variable problem is

$$\sum_{r=1}^{n} {}^{n}C_r = 2^n - 1$$

The exhaustive partitioning is normally not required however

$$\text{since if} \quad F_i(X) = \theta \quad \text{then} \quad \left. \begin{array}{l} F_{i,j}(X) = \theta \,, \\ F_{i,j,k}(X) = \theta \quad \text{etc.} \end{array} \right\} \quad \ldots (1.39)$$

$$\text{Similarly if} \quad F_{i,j}(X) = \theta \quad \text{then} \quad F_{(i,j),k}(X) = \theta \quad \text{etc.} \quad \ldots (1.40)$$

$$\text{Also if} \quad P_a = \theta \quad \text{then} \quad P_{(a+b)} = \theta \quad , \; a \leqslant (a+b) \leqslant n \text{ . etc,} \quad (1.41)$$

$$\text{and if} \quad F_i(X) \; F_j(X) = \theta \quad \text{then} \quad F_{i,j}(X) = \theta \quad \text{etc.} \quad \ldots (1.42)$$

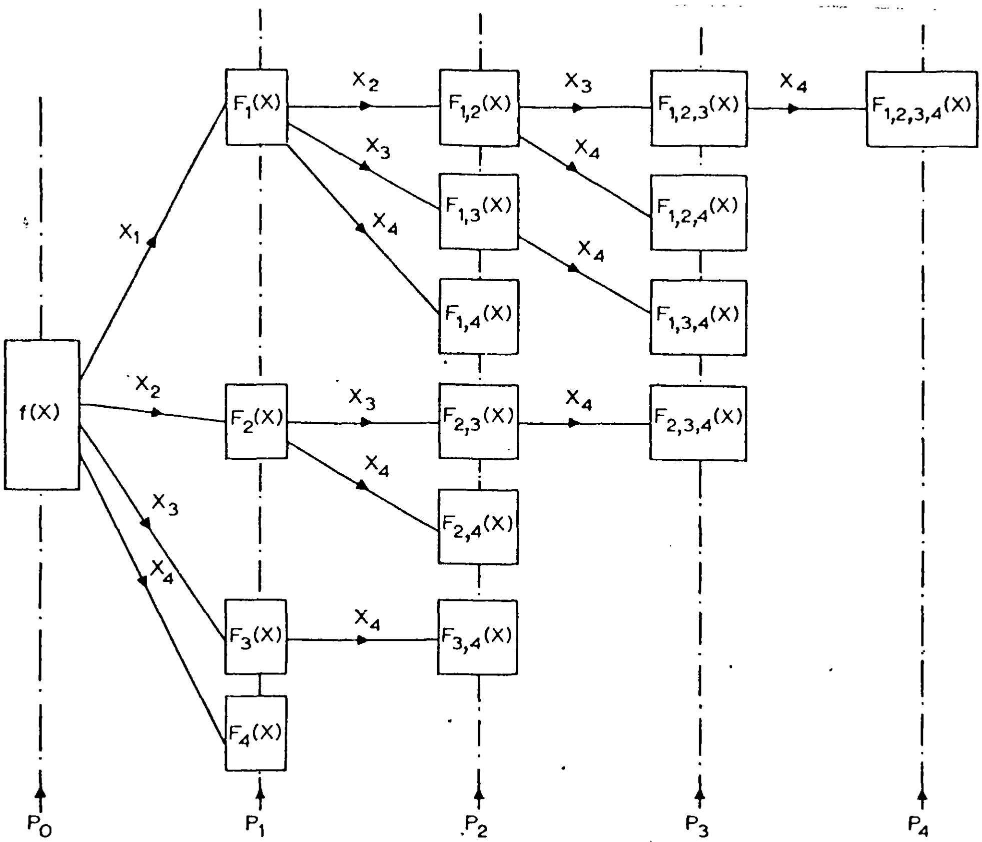
The number of variables in which partition and re-partition is possible is therefore limited from the beginning.

An example of the extraction of the prime implicants of a third -order function is now given.

Consider the function shown in the table below. Let $F(X) \triangleq P_0$

Now if $F_1(X), F_2(X), F_3(X)$ are derived as described above, inspection shows that none yield a null set, ie. they are all re-partitionable. (See table P76)

P_1 is evaluated from $P_1 = \bigcup_{i=1}^{3} F_i(X)$, and then $P_0 \cap \bar{P}_1$ yields a null set. This means that all true minterms of the function are adjacent in at least one variable, none being irreducible.



Exhaustive Partitioning Map

for a Fourth-Order Function

Fig. 20

Similarly if $F_{1,2}(X)$, $F_{1,3}(X)$, $F_{2,3}(X)$ and P_2 are derived it can be seen that $P_1 \cap \bar{P}_2$ yields a true minterm at n-tuple 7 , which must be adjacent in one variable only. Inspection of $F_2(X)$ shows that the minterm at n-tuple 7 is adjacent to minterm 5 in variable number 2.

The only remaining partition possible is $F_{1,2,3}(X)$ which must be a null set since $F_{1,2}(X) = \theta$, see equation (1.40). Thence $P_3 = \theta$ and $P_2 \cap \bar{P}_3 = 0,1,4,5$. The minterms at these n-tuples are adjacent in variables 1 and 3 from $F_{1,3}(X)$.

The function may thus be expressed as :

$$F(X) = (5,7)/(0,1,4,5)$$

TABLE.

	0	1	2	3	4	5	6	7	n-tuple
P_0	1	1	0	0	1	1	0	1	$F(X)$
	1	1	0	0	1	1	0	0	$F_1(X)$
	0	0	0	0	0	1	0	1	$F_2(X)$
	0	0	0	0	1	1	0	0	$F_3(X)$
P_1	1	1	0	0	1	1	0	1	$P_0 \cap \bar{P}_1 = 0$
	0	0	0	0	0	0	0	0	$F_{1,2}(X)$
	1	1	0	0	1	1	0	0	$F_{1,3}(X)$
	0	0	0	0	0	0	0	0	$F_{2,3}(X)$
P_2	1	1	0	0	1	1	0	0	$P_1 \cap \bar{P}_2 = 7$
P_3	0	0	0	0	0	0	0	0	$P_2 \cap \bar{P}_3 = 0,1,4,5$

Note $F_{1,2,3}(X) = \theta$

The Karnaugh map corresponding to this result is :

$x_3 \backslash x_1,x_2$	00	01	11	10
0	1	0	0	1
1	1	0	1	1

This method has the following potential advantages over the method of Quine-McCluskey :

1/ The intermediate results comprise vectors of known dimension whereas the Quine-McCluskey method generates tables of indeterminate size. The storage of intermediate data is thus simplified which is important when computer implementation is considered (using non-dynamic storage programmes).

2/ Because of its simple and recursive nature the Boolean matrix method of prime-implicant extraction is to be preferred from a programming viewpoint.

3/ The simultaneous extraction of prime-implicants of several functions is possible which , together with the restrictions on re-partitioning given in equations (1.39 - 1.42) , makes the matrix method more efficient than that of Quine-McCluskey.

4/ The dependence of a function , or functions , upon particular variables may be determined without recourse to the evaluation of all possible partitions using the matrix method.

See also reference 11.

1.5.7 Logic Synthesis by Iterative methods.

It has been shown that a logic system specified by $\left[D\right]$ may be represented as :

$$\left[B\right]\left[C\right]\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \left[D\right]\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} .$$

In addition if $\left[C\right]$ is known in this equation then

$\left[B\right] = \left[D\right]\left[C\right]^{-1}$ may be evaluated providing that the original equation is valid .

Now it follows that if $\left[C\right]$ represents a logic module of the type available to synthesise $\left[D\right]$, it is possible to determine if in fact $\left[C\right]$ may be used in the synthesis of $\left[D\right]$ by establishing the validity of the above equation. If $\left[C\right]$ does satisfy $\left[D\right]$ then $\left[B\right]$ may be evaluated and represents the remainder of the system to be synthesised. $\left[D\right]$ may then be copied into $\left[B\right]$ and the procedure repeated until $\left[B\right]$ is found to be equal to the unit matrix . The system is then synthesised.

In many cases $\left[C\right]$ will represent particular configurations of NAND,NOR,EX-OR gates , but in general there is no restriction on the type of module that $\left[C\right]$ may represent.

In its simplest form this synthesis algorithm gives rise to an iterative procedure which does not afford optimisation except on a comprehensive search basis. In this respect the method is similar to that of Roth and Ashenhurst , see references 12 and 13. It differs from the methods of Roth and Ashenhurst however in that multiple-output systems may be synthesised without resort to special techniques.

In order that this algorithm may be executed with maximum efficiency on the digital computer it is advantagous to employ an implementation that avoids the generation of the intermediate results arising from the application of Criterion 1 and the evaluation of $\left[C\right]^{-1}$.

The method illustrated in the following example is proposed.

Consider $\left[B\right]\left[C\right]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \left[D\right]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ where

$$\left[B\right]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & * & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} .$$

Let $\left[B\right]$ be filled initially with don't care states :

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & * & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

, or in decimal form

$$\begin{bmatrix} * & * & * & * \end{bmatrix}\begin{bmatrix} 3 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 & \frac{1}{3} & 1 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Now execute the following trial multiplication :

The first column of [C] has the value '3' , therefore the value of [B] at n-tuple 3 must take the value of the first column of [D] :

$$\begin{array}{c}\begin{bmatrix} * & * & * & 3\end{bmatrix}\\ \begin{matrix}0&1&2&3\end{matrix}\end{array}\begin{bmatrix}3&1&1&0\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] = \begin{bmatrix}3&\frac{1}{3}&1&2\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] \quad ;$$

the second column of [C] has the value '1' ; therefore the value of [B] at n-tuple 1 must take the value of the second column of [D] :

$$\begin{array}{c}\begin{bmatrix} * & \frac{1}{3} & * & 3\end{bmatrix}\\ \begin{matrix}0&1&2&3\end{matrix}\end{array}\begin{bmatrix}3&1&1&0\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] = \begin{bmatrix}3&\frac{1}{3}&1&2\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] \quad ;$$

the third column of [C] has the value '1' therefore the value of [B] at n-tuple 1 must take the value of the third column of [D]. Since the value of [B] has already been established as '$\frac{1}{3}$' it is necessary to check if the value now proposed is compatible.

ie. is '$\frac{1}{3}$' compatible with '1' ? - or is $\left.\begin{matrix}*\\1\end{matrix}\right]$ compatible with $\left.\begin{matrix}0\\1\end{matrix}\right]$?

Clearly these two values are compatible only if $\left.\begin{matrix}*\\1\end{matrix}\right]$ in both [B] and [D] take the value $\left.\begin{matrix}0\\1\end{matrix}\right]$.

Then

$$\begin{array}{c}\begin{bmatrix} * & 1 & * & 3\end{bmatrix}\\ \begin{matrix}0&1&2&3\end{matrix}\end{array}\begin{bmatrix}3&1&1&0\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] = \begin{bmatrix}3&1&1&2\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] \quad .$$

Finally the fourth column of [C] has the value '0' therfore the value of [B] at n-tuple 0 must take the value of the fourth column of [D] :

$$\begin{array}{c}\begin{bmatrix} 2 & 1 & * & 3\end{bmatrix}\\ \begin{matrix}0&1&2&3\end{matrix}\end{array}\begin{bmatrix}3&1&1&0\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] = \begin{bmatrix}3&1&1&2\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right]$$

The module [C] can thus be used to synthesise [D] , the function remaining to be synthesised being

$$\begin{bmatrix}1&0&*&1\\0&1&*&1\end{bmatrix}\left.\begin{matrix}x_1\\x_2\end{matrix}\right] .$$

This trial-and-error method of solving $\begin{bmatrix}B\end{bmatrix}\begin{bmatrix}C\end{bmatrix}\begin{bmatrix}x_1\\ \cdot\\ \cdot\\ x_n\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}\begin{bmatrix}x_1\\ \cdot\\ \cdot\\ x_n\end{bmatrix}$

thus overcomes the problems associated with applying Criterion 1 and evaluating the inverse matrix . If the equation is not valid $\begin{bmatrix}C\end{bmatrix}$ will force $\begin{bmatrix}B\end{bmatrix}$ to take incompatible values .

eg. $\begin{bmatrix}B\end{bmatrix}\begin{bmatrix}2 & 2 & 1 & 3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}1 & 2 & 0 & 3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$ will force the

value of $\begin{bmatrix}B\end{bmatrix}$at n-tuple 2 to take the simultaneous values '1' and '2', or $\begin{bmatrix}0\\ 1\end{bmatrix}$ and $\begin{bmatrix}1\\ 0\end{bmatrix}$ which is impossible. The detection of such incompatible cases will , in general , occur before the whole trial multiplication is complete , this results in a fast procedure.

As an example of a system synthesis consider the following simple example.

A system is defined by the matrix $\begin{bmatrix}D\end{bmatrix}$ where

$$\begin{bmatrix}D\end{bmatrix} = \begin{bmatrix}3 & 3 & 2 & \genfrac{}{}{0pt}{}{2}{6} & 7 & 5 & 6 & 5\end{bmatrix} .$$

Synthesise the system using the logic module of Fig. 21a together with the comprehensive set of interconnection modules and associated matrices shown in Fig.21b - g.

Solution

Let the system be represented by

$\begin{bmatrix}B\end{bmatrix}\begin{bmatrix}C\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$, where $\begin{bmatrix}C\end{bmatrix}$ is composed of the

matrix corresponding to the given logic module post-multiplied by one of the possible interconnection modules of Fig. 21 .

For interconnection 21 b, the equation $\begin{bmatrix}B\end{bmatrix}\begin{bmatrix}C\end{bmatrix} = \begin{bmatrix}D\end{bmatrix}$ is :

$$\begin{bmatrix}B\end{bmatrix}\underbrace{\left[\begin{bmatrix}1 & 1 & 3 & 2 & 5 & 5 & 7 & 6\\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7\end{bmatrix}\begin{bmatrix}0 & 1 & 2 & 3 & 4 & 5 & 6 & 7\end{bmatrix}\right]}_{\begin{bmatrix}C\end{bmatrix}} = \begin{bmatrix}3 & 3 & 2 & \genfrac{}{}{0pt}{}{2}{6} & 7 & 5 & 6 & 5\end{bmatrix}$$

Evaluating $\begin{bmatrix}C\end{bmatrix}$ and letting $\begin{bmatrix}B\end{bmatrix}$ have initially don't care states:

$$\begin{bmatrix}* & * & * & * & * & * & * & *\end{bmatrix}\begin{bmatrix}1 & 1 & 3 & 2 & 5 & 5 & 7 & 6\end{bmatrix} = \begin{bmatrix}3 & 3 & 2 & \genfrac{}{}{0pt}{}{2}{6} & 7 & 5 & 6 & 5\end{bmatrix}$$

Matrix.	Module.	
$\begin{bmatrix}1 & 1 & 3 & 2 & 5 & 5 & 7 & 6\end{bmatrix}$		21a
$\begin{bmatrix}0 & 1 & 2 & 3 & 4 & 5 & 6 & 7\end{bmatrix}$		21b
$\begin{bmatrix}0 & 2 & 1 & 3 & 4 & 6 & 5 & 7\end{bmatrix}$		21c
$\begin{bmatrix}0 & 1 & 4 & 5 & 2 & 3 & 6 & 7\end{bmatrix}$		21d
$\begin{bmatrix}0 & 2 & 4 & 6 & 1 & 3 & 5 & 7\end{bmatrix}$		21e
$\begin{bmatrix}0 & 4 & 1 & 5 & 2 & 6 & 3 & 7\end{bmatrix}$		21f
$\begin{bmatrix}0 & 4 & 2 & 6 & 1 & 5 & 3 & 7\end{bmatrix}$		21g

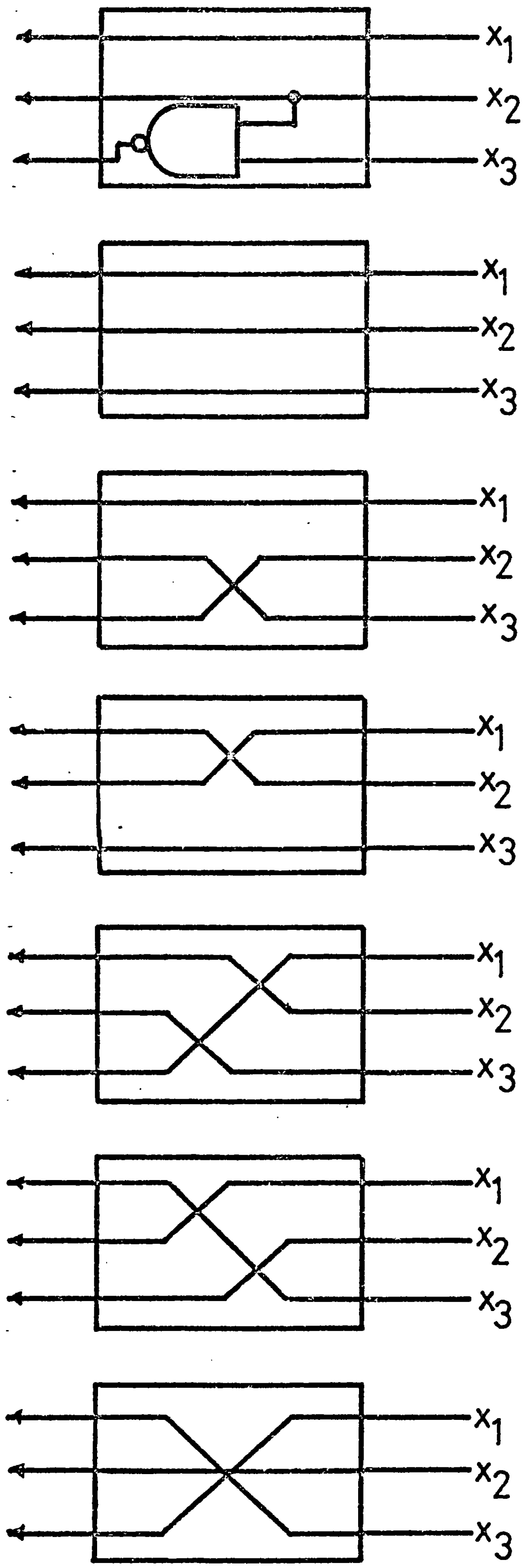


Fig. 21.

Carrying out the trial multiplication described above :

$$\underset{0\;1\;2\;3\;4\;5\;6\;7}{[*\;3\;\tfrac{2}{6}\;2\;*\;7\;*\;*]}[1\;1\;3\;2\;5\;5\;7\;6] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

the multiplication fails at the point shown. The equation is not valid.

Try interconnection 21c :

$$[B]\left[\underset{0\;1\;2\;3\;4\;5\;6\;7}{[1\;1\;3\;2\;5\;5\;7\;6]}[0\;2\;1\;3\;4\;6\;5\;7]\right] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

then

$$\underset{0\;1\;2\;3\;4\;5\;6\;7}{[*\;3\;*\;3\;*\;*\;*\;*]}[1\;3\;1\;2\;5\;7\;5\;6] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

fails at point shown.

Try interconnection 21d

$$[B]\left[\underset{0\;1\;2\;3\;4\;5\;6\;7}{[1\;1\;3\;2\;5\;5\;7\;6]}[0\;1\;4\;5\;2\;3\;6\;7]\right] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

then

$$[*\;3\;5\;7\;*\;2\;5\;6][1\;1\;5\;5\;3\;2\;7\;6] = [3\;3\;2\;2\;7\;5\;6\;5]$$

<u>gives a solution</u>. Note that $[D]$ is restricted as shown .

Try interconnection 21e

$$[B]\left[\underset{0\;1\;2\;3\;4\;5\;6\;7}{[1\;1\;3\;2\;5\;5\;7\;6]}[0\;2\;4\;6\;1\;3\;5\;7]\right] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

then

$$\underset{0\;1\;2\;3\;4\;5\;6\;7}{[*\;3\;*\;3\;*\;2\;*\;\tfrac{2}{6}]}[1\;3\;5\;7\;1\;2\;5\;6] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

fails at point shown.

Try interconnection 21f

$$[B]\left[\underset{0\;1\;2\;3\;4\;5\;6\;7}{[1\;1\;3\;2\;5\;5\;7\;6]}[0\;4\;1\;5\;2\;6\;3\;7]\right] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

then

$$\underset{0\;1\;2\;3\;4\;5\;6\;7}{[*\;3\;*\;*\;*\;3\;*\;*]}[1\;5\;1\;5\;3\;7\;2\;6] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

fails at point shown.

Try interconnection 21g

$$[B]\left[\underset{0\;1\;2\;3\;4\;5\;6\;7}{[1\;1\;3\;2\;5\;5\;7\;6]}[0\;4\;2\;6\;1\;5\;3\;7]\right] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

then

$$\underset{0\;1\;2\;3\;4\;5\;6\;7}{[*\;3\;*\;2\;*\;3\;*\;\tfrac{2}{6}]}[1\;5\;3\;7\;1\;5\;2\;6] = [3\;3\;2\;\tfrac{2}{6}\;7\;5\;6\;5]$$

fails at point shown.

All interconnection possibilities have been tried giving only one solution , ie. $[B] = [*\;3\;5\;7\;*\;2\;5\;6]$, and since $[B]$ is singular it cannot be a pertubation of the $[A]$ matrix. A second stage of synthesis is therefore necessary. The implementation of the first stage of synthesis is given by Fig. 22 a .

To evaluate the second stage of synthesis the remainder $[B]$ is copied into $[D]$ and the process is repeated.

Then $[B][C] = [*\ 3\ 5\ 7\ *\ 2\ 5\ 6]$

Try interconnection 21b

$$[B]\left[\begin{array}{c}[1\ 1\ 3\ 2\ 5\ 5\ 7\ 6]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7]\right] = [*\ 3\ 5\ 7\ *\ 2\ 5\ 6]$$

then

$$\begin{array}{c}[*\ 3\ 5\ 7\ *\ 2\ 6\ 5]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[1\ 1\ 3\ 2\ 5\ 5\ 7\ 6] = [3\ 3\ 5\ 7\ 2\ 2\ 5\ 6]$$

<u>gives a solution</u> . Note that $[D]$ is restricted as shown.

Since $[B]$ is singular it cannot be a pertubation of $[A]$.

Try interconnection 21c

$$[B]\left[\begin{array}{c}[1\ 1\ 3\ 2\ 5\ 5\ 7\ 6]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[0\ 2\ 1\ 3\ 4\ 6\ 5\ 7]\right] = [*\ 3\ 5\ 7\ *\ 2\ 5\ 6]$$

then

$$\begin{array}{c}[*\ 5\ 7\ 3\ *\ 5\ 6\ 2]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[1\ 3\ 1\ 2\ 5\ 7\ 5\ 6] = [5\ 3\ 5\ 7\ 5\ 2\ 5\ 6]$$

<u>gives a solution</u>. Note that $[D]$ is restricted as shown .

Since $[B]$ is singular it cannot be a pertubation of $[A]$.

Try interconnection 21d

$$[B]\left[\begin{array}{c}[1\ 1\ 3\ 2\ 5\ 5\ 7\ 6]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[0\ 1\ 4\ 5\ 2\ 3\ 6\ 7]\right] = [*\ 3\ 5\ 7\ *\ 2\ 5\ 6]$$

then

$$\begin{array}{c}[*\ 3\ *\ *\ *\ 5\ *\ *]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[1\ 1\ 5\ 5\ 3\ 2\ 7\ 6] = [3\ 3\ 5\ 7\ *\ 2\ 5\ 6]$$

fails at point shown.

Try interconnection 21e

$$[B]\left[\begin{array}{c}[1\ 1\ 3\ 2\ 5\ 5\ 7\ 6]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[0\ 2\ 4\ 6\ 1\ 3\ 5\ 7]\right] = [*\ 3\ 5\ 7\ *\ 2\ 5\ 6]$$

then

$$\begin{array}{c}[*\ *\ 2\ 3\ *\ 5\ 6\ 7]\\ \mathbf{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\end{array}[1\ 3\ 5\ 7\ 1\ 2\ 5\ 6] = [*\ 3\ 5\ 7\ *\ 2\ 5\ 6]$$

<u>gives a solution</u> . Moreover $[B]$ is non-singular and can take the form of the $[A]$ matrix . The synthesis is therefore completed using this interconnection.

The complete synthesis is shown incircuit form in Fig. 22b.

A test programme, written in Fortran IV / Machine code,has been run successfully for the above algorithm employing gates of the NAND, NOR,EX-OR type for problems of up to fifth order.

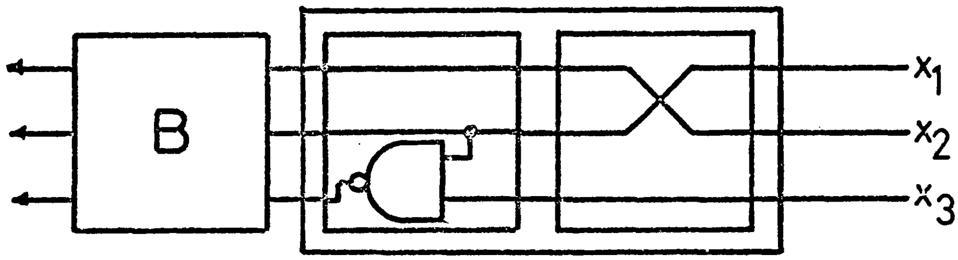
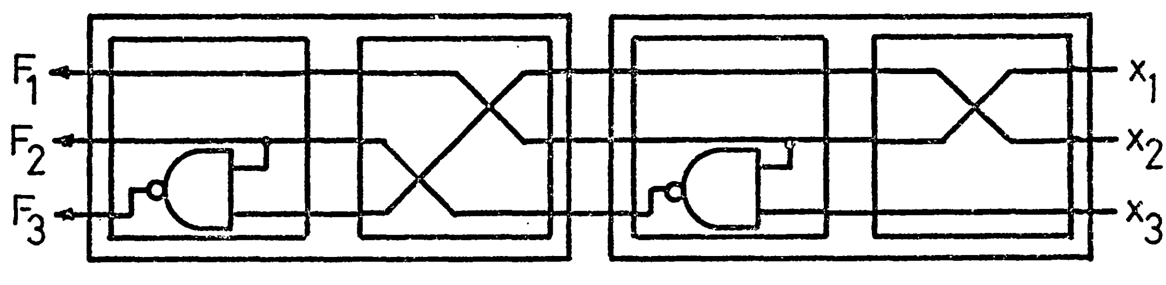


Fig. 22a



$$\begin{bmatrix} 0&0&0&0&1&1&1&1 \\ 1&1&1&1&1&0&1&0 \\ 1&1&0&0&1&1&0&1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

Fig. 22b

In practice it has been found advantagous to search for disjunctive decompositions initially and , if none are found , proceed with the search for non-disjunctive decompositions.

See also reference 14 .

The abovementioned computer programme is able to find all possible disjunctive and non-disjunctive decompositions for a fifth-order system , using up to three input NAND/NOR gates , in approximately 2 seconds for each stage of synthesis. The storage required (P.D.P.8E) is 11901 K/words.

1.6 Conclusions.

It has been demonstrated that Boolean matrices , of the type defined , enable cascaded , multi-output logic modules to be both described , in terms of functional capability and topology , and manipulated. It has been shown that the algebra associated with these matrices is capable of analysing and synthesising such systems even where unspecified conditions (don't care states) are involved in the system description. The algebra is also able to define dependent functions ; the full implications of this are not yet known.

Two novel methods of logic circuit synthesis have been described which follow naturally from the consideration of 'Boolean matrix operators' and 'valid equations' . The first of these enables the dependence of a function upon any chosen set of its defining variables to be determined. It has been shown that the exhaustive implementation of this technique , using Boolean matrices , enables the prime implicants of several functions to be extracted simultaneously. This method is an attractive alternative to that of Quine-McCluskey. The second synthesis method arises from the consideration of 'valid equations' and the 'inverse singular matrix' . It is an iterative technique which , on an exhaustive search basis , enables optimum syntheses of multi-output systems to be found. Again these systems may be partially specified. Both of these synthesis methods , particularly the latter , are especially easy to implement using the digital computer.

Several iterative synthesis procedures , of various types , have been published in recent years.† It is felt that the method described herein probably represents the most effective <u>simple</u> multi-output synthesis to date.

The main disadvantage of iterative techniques is that they

† See for example Ref. 12

are relatively slow to 'converge' to an optimum solution , especially when the number of defining variables is large. (In this respect the method developed in this chapter is no exception.) Moreover the expertise of the logic designer can play little or no part in their execution. (In the author's opinion the rather unsucessful attempts to introduce 'heuristics' into such methods is an attempt to do this.) At this point in the research therefore, a search was instigated for possible techniques which would a) generate an acceptable synthesis very quickly ,and b) enable the logic designer to assimilate the pertinent features of the system to be designed very easily and to be able to act on this information. At present the best method of evaluating the properties of a Boolean function quickly is with the aid of a Karnaugh map. This method however is of limited value when the number of defining variables is large.

The result of the search for a new method of interpreting Boolean functions according to the above criteria appear in Chapter 2.

CHAPTER 2.

The Application of the Rademacher/Walsh Transform to Logic Design and Boolean Function Classification.

2.1 Introduction.

In 1922 Rademacher published a new set of orthogonal functions taking the value $\pm$ 1 in the interval (0,1), see reference 15 . This set of functions however was incomplete - a finite set of such functions does not form a sub-group.

Working independently, in 1923 , Walsh published a set of orthogonal functions taking the value $\pm$ 1 in the interval (0,1), see reference 16. The Walsh functions , in addition to forming a complete set , have the Rademacher functions as a generating set. That is to say , any set of Walsh functions may be generated from a suitable set of Rademacher functions. See also references 17 ,18.

Because the Walsh functions have properties analogous to trigonometric functions , considerable research has gone into employing 'Walsh waves' for the transmission of sampled-data digital information. Other areas of application have been in the fields of signal filtering and pattern recognition.

In the field of logic design the Walsh functions appear to have been employed relatively little. Chow , reference 19, showed that certain parameters were sufficient to characterise threshold functions and Dertouzos, reference 20 , showed that these parameters were in fact Walsh transform coefficients. Dertouzos also developed operators for the manipulation of these coefficients to facilitate threshold logic synthesis. (It is largely an extention of the work of Dertouzos that will be considered here.) In addition Ito, reference 21 , has considered the application of Walsh functions to the recognition of binary-valued functions on a statistical basis. Hurst, reference 22 , has considered the general possibilities of the application of Walsh functions to the synthesis of binary functions both in terms of threshold and conventional logic circuitry.

The justification for the analysis of Boolean functions under the Rademacher/Walsh transform lies in the fact that certain Boolean operations may be executed more easily in the transform domain and that many of the properties of Boolean functions which are normally difficult to determine , eg. linear separability , are best characterised in this domain. In this respect an analogy can be drawn between this transform and the Fourier transform.

It is the purpose of this chapter to show that particular operations in the transform domain have certain properties which lend themselves naturally to the synthesis of logic functions , and to illustrate how these operations may be extended to facilitate the solution of more complex problems.

The synthesis of logic functions both in terms of threshold gates and vertex (NAND,NOR,AND,OR) gates is considered.

In addition it is shown that these operations lead to a very efficient method of classifying Boolean functions.

2.2 The Rademacher/Walsh Transform.

2.2.1 Introduction.

In this section a particular form of the above transform will be defined which has properties which are especially relevant to the field of logic synthesis. For an alternative definition of this form of the above transform see reference 23.

The more general properties of the Walsh transform may be found in references 16 and 18.

2.2.2 Definitions and Properties.

Consider the square Boolean matrix $\left[T\right]$ of Fig. 23. For reasons that will become apparent later a $2^n \times 2^n$ matrix is said to have an order n . For example in Fig.23 , $n=4$.

$$
[T] = \begin{array}{c}
t_{1,j} \\ t_{2,j} \\ t_{3,j} \\ t_{4,j} \\ t_{5,j} \\ t_{6,j} \\ t_{7,j} \\ t_{8,j} \\ t_{9,j} \\ t_{10,j} \\ t_{11,j} \\ t_{12,j} \\ t_{13,j} \\ t_{14,j} \\ t_{15,j} \\ t_{16,j}
\end{array}
\left[\begin{array}{cccccccccccccccc}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\
0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0 \\
0&0&1&1&0&0&1&1&1&1&0&0&1&1&0&0 \\
0&1&0&1&0&1&0&1&1&0&1&0&1&0&1&0 \\
0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0 \\
0&1&0&1&1&0&1&0&0&1&0&1&1&0&1&0 \\
0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0 \\
0&0&1&1&1&1&0&0&1&1&0&0&0&0&1&1 \\
0&1&0&1&1&0&1&0&1&0&1&0&0&1&0&1 \\
0&1&1&0&0&1&1&0&1&0&0&1&1&0&0&1 \\
0&1&1&0&1&0&0&1&0&1&1&0&1&0&0&1 \\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0
\end{array}\right]
\begin{array}{ll}
x_0 & \text{Zero ordered partition} \\
x_1 & \\
x_2 & \text{First order partition} \\
x_3 & \text{(Rademacher Functions)} \\
x_4 & \\
x_{12} & \\
x_{13} & \\
x_{14} & \text{Second order partition} \\
x_{23} & \\
x_{24} & \\
x_{34} & \\
x_{123} & \\
x_{124} & \text{Third order partition} \\
x_{134} & \\
x_{234} & \\
x_{1234} & \text{Fourth order partition}
\end{array}
$$

$$
\left[\begin{array}{cccccccccccccccc}
0&1&1&0&1&0&0&1&0&0&0&1&1&1&1&0
\end{array}\right] \quad \text{Sample Function}
$$

Fig. 23.

The matrix $\left[T\right]$ has , by definition , the following properties for any n :

1/ The members of the first row of $\left[T\right]$ are equal to zero.

$$t_{1,j} \triangleq 0 \quad , \quad 1 \leqslant j \leqslant 2^n \qquad \ldots (2.1)$$

2/ The second to (n+1) th rows have the property

$$\left.\begin{array}{l} t_{i,j} \triangleq 1 \text{ when } \left\{(j-1) \text{ modulo } (2^{n-i+2})\right\} \geqslant 2^{n-i+1} \\ t_{i,j} \triangleq 0 \text{ otherwise.} \quad 2 \leqslant i \leqslant (n+1) \quad , \quad 1 \leqslant j \leqslant 2^n . \end{array}\right\} \ldots (2.2)$$

These are the Rademacher functions , reference 15 , with range 0,1 .

3/ The remaining (n+2) to 2^n rows are equal to all possible combinations of the exclusive-OR 's of rows 2 to (n+1) of $\left[T\right]$ taken one-at-a-time two-at-a-time. . . . n at-a-time. These combinations are taken in ascending order , ie. in Fig. 23, where n=4 : $t_{6,j} = (t_{2,j} \oplus t_{3,j})$, $t_{7,j} = (t_{2,j} \oplus t_{4,j})$ $t_{8,j} = (t_{2,j} \oplus t_{5,j})$, . . . , $t_{11,j} = (t_{4,j} \oplus t_{5,j})$, $t_{12,j} = (t_{2,j} \oplus t_{3,j} \oplus t_{4,j})$ . . . etc.

The complete set of functions defined above are the Walsh functions* in the range 0,1 .

---

* Originally Walsh defined these functions in the range 1,-1 . It is convenient for the applications to be considered to replace the value 1 in the range 1,-1 by 0 and to replace the value -1 in the range 1,-1 by 1 . This gives the Walsh functions in the range 0,1 defined above. Although it is convenient to develop logic synthesis theory using the Walsh functions in the range 0,1 in practice the transformation operation dscribed on page 94 is carried out in the range 1,-1 for reasons of computational speed. See also reference 24.

It is a property of the Rademacher functions in the range 0,1 that any Boolean function may be defined upon them. For example in Fig. 23, where $n=4$, the set of column vectors of the Rademacher functions constitute the set of n-tuples of a fourth order function.

In general the j th n-tuple of an n th order function may be defined as :

$$\psi_j, \quad j = \sum_{i=2}^{n+1} 2^{(n+2-i)} \times t_{i,j} \qquad \ldots \quad (2.3)$$

It is therefore possible to label each of the Rademacher functions as defining variables in the same way as in a truth table ; namely

the rows of $[T]$, $t_{i,j}$, $\left.\begin{matrix} 2 \leqslant i \leqslant (n+1) \\ 1 \leqslant j \leqslant 2^n. \end{matrix}\right\}$ are labelled x_{i-1}

the rows $t_{i,j}$, $\left.\begin{matrix} (n+2) \leqslant i \leqslant 2^n \\ 1 \leqslant j \leqslant 2^n \end{matrix}\right\}$ are labelled as

$x_{1,2}$, $x_{1,3}$, . . . , $x_{(n-1),n}$, $x_{1,2,3}$, . . . , $x_{(n-2),(n-1),n}$ . . etc.

Where $x_{1,2}$ denotes $x_1 \oplus x_2$ etc. This labelling follows from the definitions given in 3/ above. An example of this labelling for a fourth order function is given in Fig. 23.

The row of $[T]$, $t_{1,j}$, $1 \leqslant j \leqslant 2^n$ is labelled, by convention, as x_0.

The matrix $[T]$ has thus been partitioned row-wise into several areas.

Now :

1/ The first row, having the subscript of x as a 0, will be called the zero-ordered partition.

2/ The second to (n+1) rows, having a single subscript, will be called the first -order partition.

3/ The remaining rows, having in ascending order, q subscripts, will be called the q th order partitions.

This particular method of row ordering has been chosen to best illustrate the use of the transform matrix $\left[T\right]$ in the field of logic design.

The definition of the transform operation is as follows :

$$\left.\begin{array}{l} r_i \triangleq 2^n - 2\left\{ \sum_{j=1}^{2^n} t_{i,j} \oplus F_{j-1}(x_1, x_2, \ldots, x_n) \right\} , \\ \qquad\qquad\qquad\qquad 1 \leqslant i \leqslant 2^n \quad . \end{array}\right\} \ldots (2.4)$$

where $\sum$ denotes arithmetic summation , and $\oplus$ denotes the exclusive-OR operator.

It can be shown that r_i under this definition can be simply stated as :

$\{$The number of agreements between row i of $\left[T\right]$ and the function $F(x_1, x_2, \ldots, x_n)\}$ - $\{$the number of disagreements between row i of $\left[T\right]$ and the function $F(x_1, x_2, \ldots, x_n)\}$

In order that the value r_i may be related to the corresponding row labelled x_s , where s represents the subscript given to the i th row of $\left[T\right]$, r_i will be labelled R_s. For example in Fig. 23

$r_6 \equiv R_{12}$, $r_{16} \equiv R_{1234}$ etc.

Under this transformation the sample Boolean function shown in Fig. 23 transforms to the vector :

0	0	4	0	0	-4	0	0
R_0	R_1	R_2	R_3	R_4	R_{12}	R_{13}	R_{14}
4	4	0	-4	-4	0	4	12
R_{23}	R_{24}	R_{34}	R_{123}	R_{124}	R_{134}	R_{234}	R_{1234}

It can be shown that the Rademacher/Walsh transform may be executed in the range -1,+1 instead of the range 0,1 as above.

Specifically if the Boolean value 1 is replaced in T above by -1 and the Boolean value 0 is replaced by +1 , the transform operation may be accomplished by simple matrix multiplication. Equation (2.4) then becomes :

$$r_i^* \triangleq \sum_{j=1}^{n}\left\{t_{i,j}^* \times F_{\psi_{j-1}}^*(x_1,x_2,\ldots,x_n)\right\}, \quad 1 \leqslant i \leqslant 2^n \quad \ldots \quad (2.5)$$

where $t_{i,j}^*$ refers to a member of $\left[T\right]$,$\left[T\right]$ being defined in the range $-1,+1$. Fig. 25 shows the sample function of Fig.23 and Fig. 24 transformed in this way.

It has been pointed out that this transform is in some ways analogous to the Fourier transform , see reference 20. In particular it is noted that the zero-ordered coefficient R_0 is in a sense a 'd.c' term in that it is a measure of the number of false minterms of the function $F(x_1,x_2,\ldots,x_n)$. The first -ordered transform coefficients $R_1,R_2,\ldots,R_n$ are a measure of the dependence of the function on the defining variables $x_1,x_2,\ldots,x_n$. The second-order transform coefficients $R_{12},R_{13},\ldots,R_{(n-1),n}$ are a measure of the dependence of the function upon $x_1 \oplus x_2$, $x_1 \oplus x_3$, $\ldots, x_{(n-1)} \oplus x_n$ etc.

For these reasons the transform coefficients will be called 'spectral coefficients' of relevant order. For example R_{12} is a second order spectral coefficient , R_{234} is a third order spectral coefficient , and so on.

To gain some insight into the composition of a Boolean function which is characterised by a particular spectral coefficient, reference should be made to Appendix 1, where the Boolean functions corresponding to the 2^n rows of $\left[T\right]$ (in the range 0,1) are plotted on Karnaugh maps for n=4.

It is important to note that the distribution of true minterms of any function in any variable , say x_1 , (that is the number of true minterms lying in x_1 , and the number lying in $\bar{x}_1$) can be

j	$j-1$	x_1	x_2	x_3	x_4	$F_{\varphi_{j-1}}(x_1,x_2,x_3,x_4)$
1	0	0	0	0	0	0
2	1	0	0	0	1	1
3	2	0	0	1	0	1
4	3	0	0	1	1	0
5	4	0	1	0	0	1
6	5	0	1	0	1	0
7	6	0	1	1	0	0
8	7	0	1	1	1	1
9	8	1	0	0	0	0
10	9	1	0	0	1	0
11	10	1	0	1	0	0
12	11	1	0	1	1	1
13	12	1	1	0	0	1
14	13	1	1	0	1	1
15	14	1	1	1	0	1
16	15	1	1	1	1	0

Truth Table Representation of the Sample Function of Fig. 23.

Fig. 24

$$\begin{bmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&-1&-1&-1&-1&-1&-1&-1&-1\\
1&1&1&1&-1&-1&-1&-1&1&1&1&1&-1&-1&-1&-1\\
1&1&-1&-1&1&1&-1&-1&1&1&-1&-1&1&1&-1&-1\\
1&-1&1&-1&1&-1&1&-1&1&-1&1&-1&1&-1&1&-1\\
1&1&1&1&-1&-1&-1&-1&-1&-1&-1&-1&1&1&1&1\\
1&1&-1&-1&1&1&-1&-1&-1&-1&1&1&-1&-1&1&1\\
1&-1&1&-1&1&-1&1&-1&-1&1&-1&1&-1&1&-1&1\\
1&1&-1&-1&-1&-1&1&1&1&1&-1&-1&-1&-1&1&1\\
1&-1&1&-1&-1&1&-1&1&1&-1&1&-1&-1&1&-1&1\\
1&-1&-1&1&1&-1&-1&1&1&-1&-1&1&1&-1&-1&1\\
1&1&-1&-1&-1&-1&1&1&-1&-1&1&1&1&1&-1&-1\\
1&-1&1&-1&-1&1&-1&1&-1&1&-1&1&1&-1&1&-1\\
1&-1&-1&1&1&-1&-1&1&-1&1&1&-1&-1&1&1&-1\\
1&-1&-1&1&-1&1&1&-1&1&-1&-1&1&-1&1&1&-1\\
1&-1&-1&1&-1&1&1&-1&-1&1&1&-1&1&-1&-1&1
\end{bmatrix}
\begin{bmatrix}1\\-1\\-1\\1\\-1\\1\\1\\-1\\1\\1\\1\\-1\\-1\\-1\\-1\\1\end{bmatrix}
=
\begin{bmatrix}0\\0\\4\\0\\0\\-4\\0\\0\\4\\4\\0\\-4\\-4\\0\\4\\12\end{bmatrix}$$

Execution of the Rademacher/Walsh Transform of the Sample Function, in the Range 1,-1.

Fig. 25

determined exactly given the value of the corresponding spectral coefficient together with the value of the zero-ordered coefficient R_0. See also Section 2.7.2.

The Rademacher/Walsh transform matrix defined in the range -1,1 has the very important property that it is orthogonal , ie

$$\left[T\right]^{-1} = \frac{1}{2^n}\left[T\right]^t \quad . \qquad \ldots \quad (2.6)$$

That is the inverse of the transform matrix $\left[T\right]$ is equal to the transpose of $\left[T\right]$ multiplied by a constant.

Because of this property algorithms can be generated which allow the transform to be executed at a much higher speed than is possible using conventional matrix multiplication. This means that it is possible to employ the techniques to be described for systems defined upon a large number of variables without undue sacrifice of computer execution time. See also reference 24.

2.3 Observations on the Significance of the Spectral Coefficients.

It was noted above that the correlation between a given Boolean function and a particular row of the transform matrix $\left[T\right]$ is given by the value of the corresponding spectral coefficient in the transform domain.

It follows therefore that a function having a relatively large positive spectral coefficient , say R_{12} , has a high correlation with $x_1 \oplus x_2$. On the other hand if the coefficient R_{12} is large and negative , the function has a high correlation with $\overline{x_1 \oplus x_2}$.

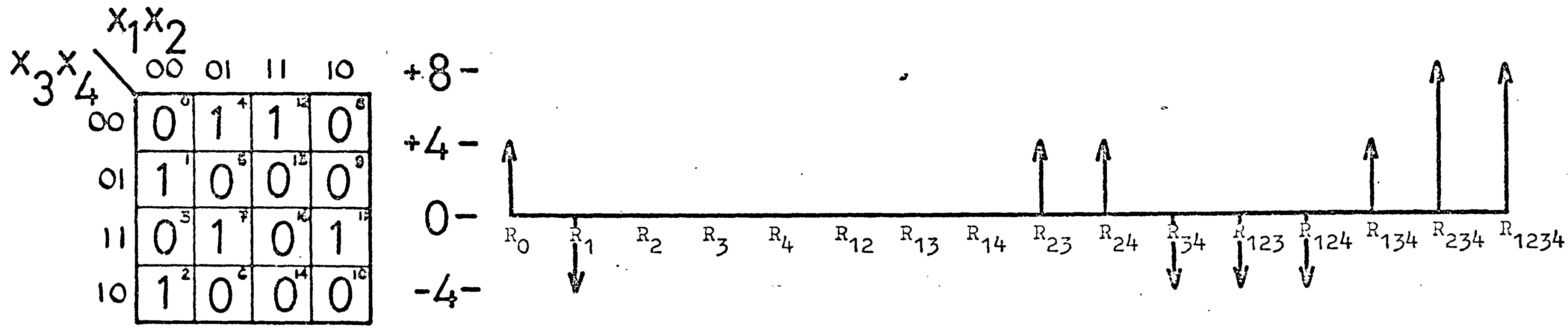
In general this interpretation may be extended to the overall distribution of the spectral coefficients in the transform domain. If, for example the function has its largest spectral coefficients in second-order positions it will be termed a 'predominantly second-ordered function ', whilst a function whose predominant spectral coefficients are high-ordered will be termed a'high-ordered' function.etc. Examples of high- and low-ordered functions appear in Fig.26a and Fig.26b. respectively. Comparisons between these functions and the Karnaugh maps of Appendix 1 is instructive. Note that the spectral coefficient R_0 does not enter into this classification as it does not contribute any information about the ordering of the function ; it is zero-ordered.

Since any Boolean function is uniquely reconstructable from its spectrum , see reference 20 , it follows that each of the spectral coefficients contain some information about the function. It has been shown that this information is not, in general , evenly distributed among the coefficients, see also reference 25. A special case is that of the linearly-separable or threshold functions, in which all the information is contained in the first (n+1) coefficients. These are the Chow parameters as shown by Dertouzos,see references 19 and20. It follows that threshold functions are predominantly first-ordered.

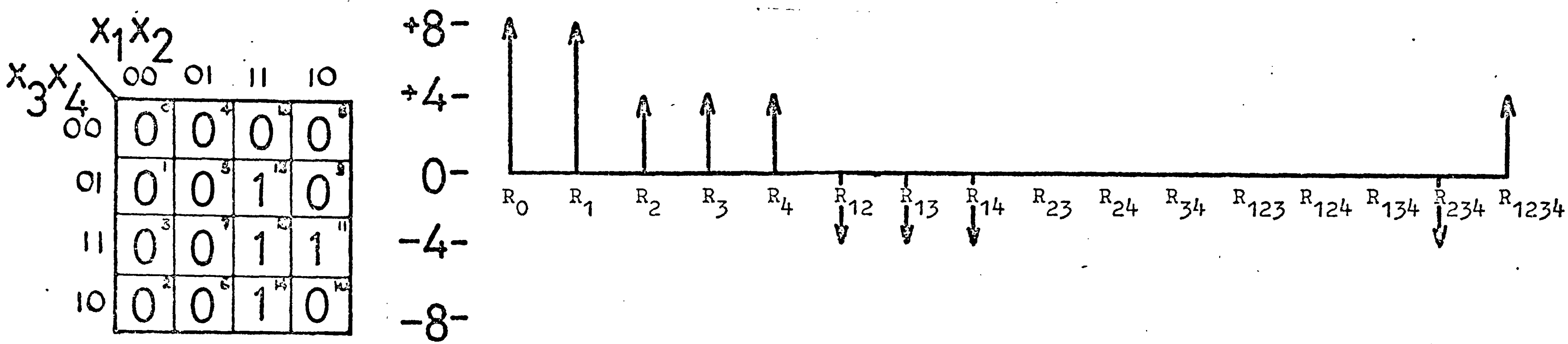
Inspection of the high-ordered function of Fig. 26a shows it to be'classically' cumbersome to synthesise from a circuit designers point of view since the true minterms of the function are scattered on the Karnaugh map and do not fall predominantly into areas corresponding to the intersection or union of any particular defining variables. The opposite is true of the low-ordered function of Fig. 26b.

These observations lead to the intuitive supposition that high-ordered functions are most easily synthesised with the aid of exclusive-OR gates . This supposition will be verified later.

In the light of the above discussion it would also appear that it is advantageous to be able to convert high-ordered functions



HIGH-ORDERED FUNCTION Fig. 26a



LOW-ORDERED FUNCTION Fig. 26b

into lower-ordered functions by some method. Such methods will be described later.

2.4 Some operations in the Transform Domain.

It is of importance to investigate the relationships between operations in the transform domain and those in the Boolean function domain, or 'Boolean domain'. By doing so it is possible to show that certain Boolean operations may be executed more easily in the transform domain and also that certain operations in the transform domain may be immediately interpreted in terms of logic circuit synthesis.

Consider the following operation :

Operation 1 .

The interchange of variables x_k with x_1 , $k \neq 1$, $k \neq 0$.

From equation (2.4)

$$\left.\begin{aligned} r_i \triangleq 2^n - 2 \left\{ \sum_{j=1}^{2^n} t_{i,j} \oplus F_{\psi_{j-1}}(x_1,x_2, \ldots ,x_n) \right\} , \\ 1 \leqslant i \leqslant 2^n . \end{aligned}\right\} \text{equn.(2.4) repeated.}$$

Substituting $x_{k,j}$ for $t_{i,j}$ and R_k for r_i in the above gives

$$\left.\begin{aligned} R_k = 2^n - 2 \left\{ \sum_{j=1}^{2^n} x_{k,j} \oplus F_{\psi_{j-1}}(x_1,x_2, \ldots ,x_n) \right\} , \\ 0 \leqslant k \leqslant n . \end{aligned}\right\} \quad \ldots \quad (2.7)$$

Define a new function

$$F'(x_1,x_2, \ldots ,x_k',x_1', \ldots ,x_n) = F(x_1,x_2, \ldots ,x_k,x_1, \ldots ,x_n) \quad \ldots \quad (2.8)$$

where $x_k' = x_1$

$k \neq 1 \neq 0$.

and $x_1' = x_k$

Then equation (2.7) can be written as

$$R'_k = 2^n - 2 \left\{ \left\{ \sum_{j=1}^{2^n} x'_{k,j} \oplus F'_{\psi_{j-1}}(x_1,x_2, \ldots ,x'_k, \ldots ,x_n) \right\} \right\}_{1 \leqslant k \leqslant n} . \quad (2.9)$$

or

$$R_1 = 2^n - 2 \left\{ \sum_{j=1}^{2^n} x_{1,j} \oplus F_{\psi_{j-1}}(x_1,x_2, \ldots ,x_1, \ldots ,x_n) \right\}_{1 \leqslant 1 \leqslant n} . \quad (2.10)$$

The equations (2.9) and (2.10) are therefore equivalent , and $R'_k = R_1$.

It can also be shown that , under this operation ,

$$R'_1 = R_k ,$$

$R'_{km} = R_{1m}$, $R'_{1m} = R_{km}$ and

, $R'_{k1} = R_{k1}$, $R'_m = R_m$, $R'_0 = R_0$ etc.

That is the resulting set of spectral coefficients $\langle R' \rangle$ are generated from $\langle R \rangle$ by replacing k by 1 in the subscripts of $\langle R \rangle$ and vice-versa.

For example if x_1 is interchanged with x_2 , the resulting spectrum $\langle R' \rangle$ is generated as ;

$R'_{13} = R_{23}$, $R'_{23} = R_{13}$ and $R'_{234} = R_{134}$, $R'_{134} = R_{234}$ etc.

It is now possible to interpret the above operation in terms of general logic circuitry.

Fig. 27a shows the implementation of the Boolean function $F(x_1,x_2, \ldots ,x_k,x_1, \ldots ,x_n)$ which has the corresponding spectrum $\langle R \rangle$.

According to the above , variables x_k and x_1 are now interchanged and a new module corresponding to $F'(x_1,x_2, \ldots ,x'_k,x'_1, \ldots ,x_n)$ is defined , as shown in Fig. 27b.

This new module has the spectrum $\langle R' \rangle$. Note that , from

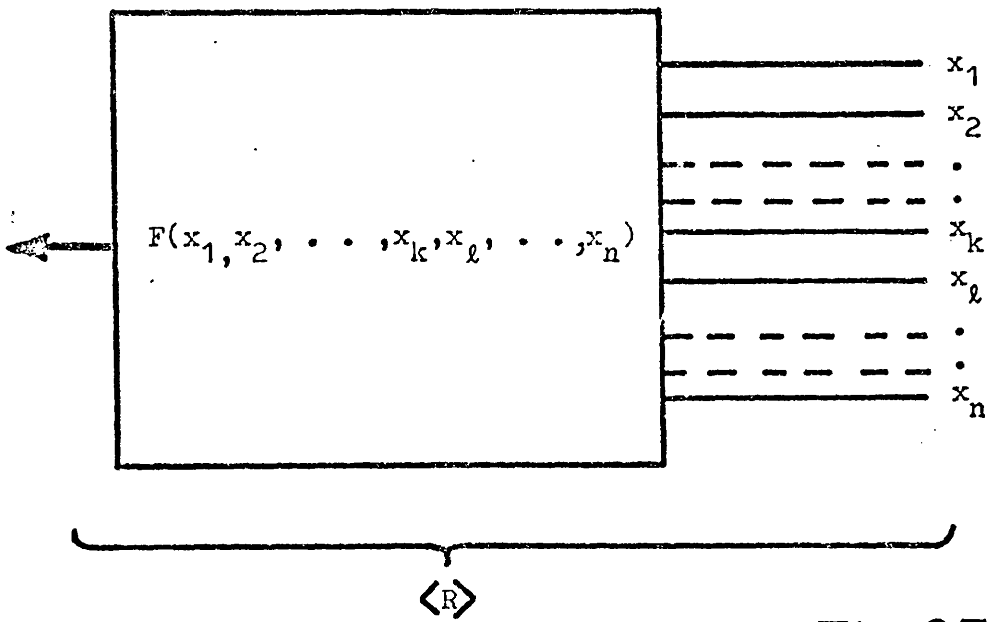


Fig.27a

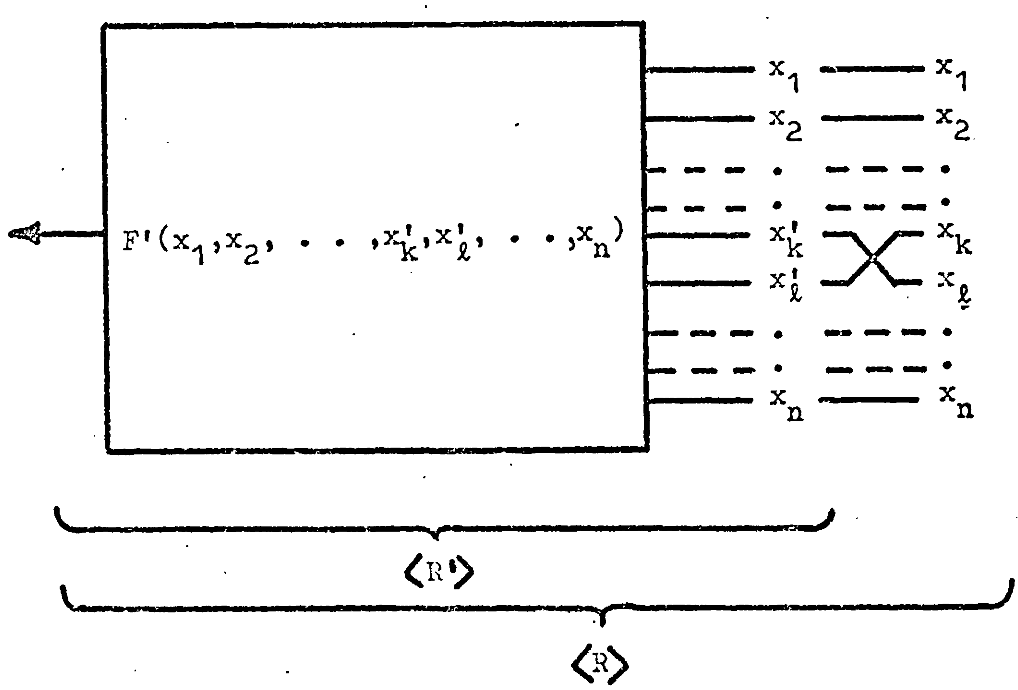


Fig. 27b

equation (2.8) , the overall transfer function of the system has not changed.

The above is an illustration of an operation in the transform domain which may be directly interpreted in terms of logic circuitry. Dertouzos , reference 20 , has considered several of these operations and the most important of these are given , without derivation , below.

Operation 1 (repeated)

Interchange of variables x_k with x_l , $k \neq l \neq 0$.

The new spectrum $\langle R' \rangle$ may be generated from the original spectrum $\langle R \rangle$ under the interchange of x_k and x_l if in $\langle R \rangle$ the subscript k is replaced by the subscript l and vice-versa.

Operation 2.

Complementation of the variable x_k : x_k' becomes $\bar{x}_k$.

The new spectrum $\langle R' \rangle$ may be generated from the original spectrum $\langle R \rangle$ under the complementation of variable x_k if in $\langle R \rangle$ the spectral coefficients having subscripts containing k are changed in sign.

Fig. 28a shows the implementation of this operation.

Operation 3.

The generation of the Dual of a function.

That is , given a function $F(x_1, \ldots, x_k, \ldots, x_n)$ having a spectrum $\langle R \rangle$ generate a function $\overline{F(\bar{x}_1, \ldots, \bar{x}_k, \ldots, \bar{x}_n)}$ having a spectrum $\langle R' \rangle$.

The new spectrum $\langle R' \rangle$ may be generated from the original spectrum $\langle R \rangle$ under the above operation if in $\langle R \rangle$ the even-ordered spectral coefficients are changed in sign. Note: R_0 is even-ordered.

Fig. 28b shows the implementation of this operation.

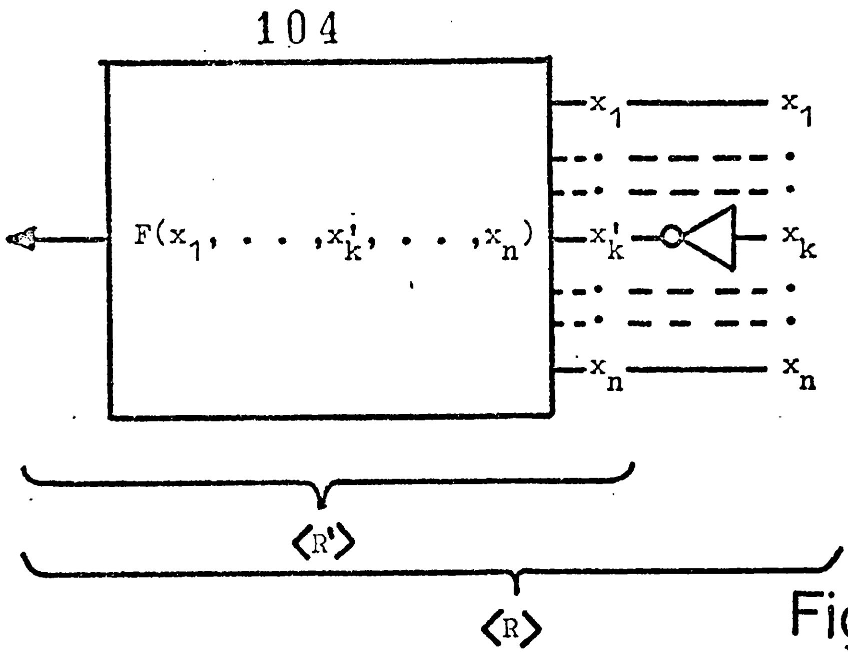


Fig. 28a

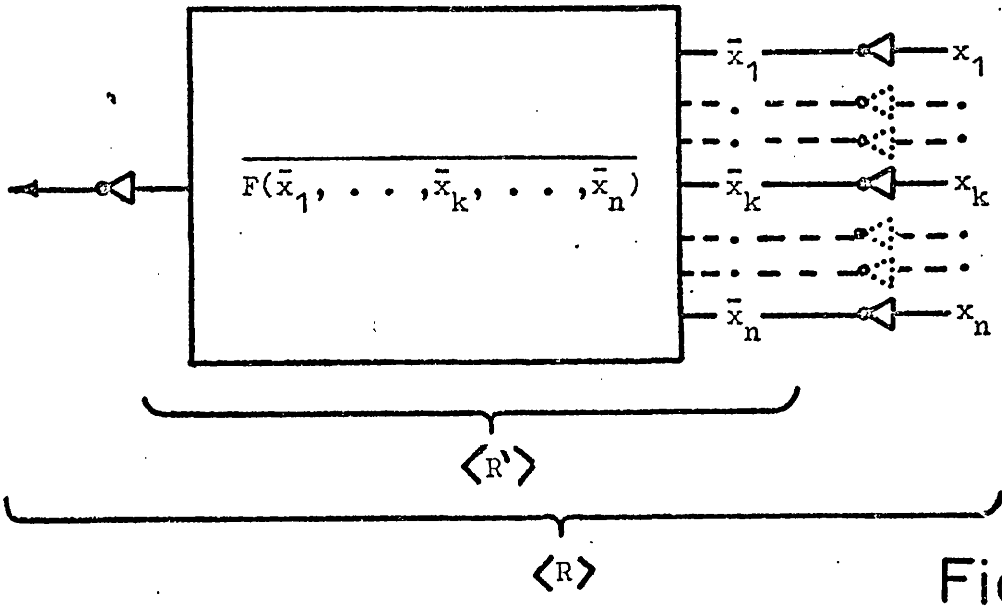


Fig. 28b

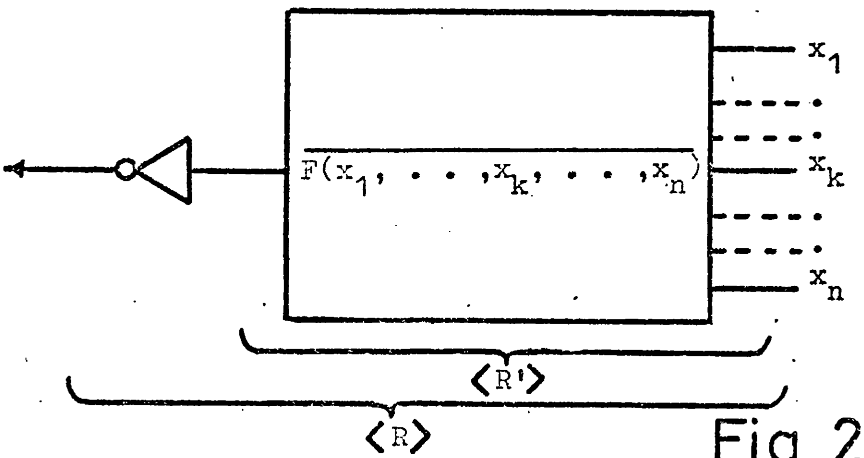


Fig. 28c

Operation 4.

The generation of the complement of a function.

That is , given the function $F(x_1, \ldots, x_k, \ldots, x_n)$ having a spectrum $\langle R \rangle$ generate a function $\overline{F(x_1, \ldots, x_k, \ldots, x_n)}$ having a spectrum $\langle R' \rangle$.

The new spectrum $\langle R' \rangle$ may be generated from the original spectrum $\langle R \rangle$ under the complementation of the function if in $\langle R \rangle$ all spectral coefficients are changed in sign.

Fig. 28c shows the implementation of this operation.

——oOo——

So far certain operations in the transform domain have been have been considered which certainly facilitate operations in the Boolean domain , but which appear to contribute little to the actual synthesis of logic functions. However Golomb , reference 26 , has shown that the ordering and complementing of the defining variables of functions enables certain functions to be classified into equivalent classes. That is , certain functions of the same order n , and which differ only in the permutation and/or complementation of their defining variables are termed equivalent. Such a classification can clearly be established by using Operations 1 and 2 .

In logic synthesis the concept of equivalent classes is important since if the synthesis of one member of such a class is known then the synthesis of any other member of the class follows by simply permutating and/or complementing the defining variables of the known system.

The number of equivalent classes is of course much smaller than the total number of functions possible , for any given n .

In order that the idea of equivalent functions may be extended it is necessary to introduce a new operation which not only facilitates logic synthesis on an equivalence basis but also finds application in the synthesis of logic functions by means of threshold gates as will be described later. This operation will be called the 'translational operation':

Operation 5.

The replacement of the defining variable x_k by $x_k \oplus x_l$, $k \neq l \neq 0$.

Recalling equation (2.4) :

$$\left.\begin{aligned} r_i &\triangleq 2^n - 2\left\{\sum_{j=1}^{2^n} t_{i,j} \oplus F_{\psi_{j-1}}(x_1,x_2,\ldots,x_n)\right\}, \\ &\qquad 1 \leqslant i \leqslant n \quad . \end{aligned}\right\} \quad \ldots (2.4) \text{ repeated.}$$

Let the given function be $F(x_1,x_2,\ldots,x_k,\ldots,x_n)$

Define a new function $F'(x_1,x_2,\ldots,x'_k,\ldots,x_n)$

$$\triangleq F(x_1,\ldots,x_k,\ldots,x_n) \quad \text{where} \quad x'_k = x_k \oplus x_l \qquad \ldots (2.11)$$

The fact that this definition gives rise to a unique new function under a basis transformation is shown in Appendix 2.

Substituting for the defining variables in equation (2.4) in the usual way gives, for the new function :

$$\left.\begin{aligned} R'_k &= 2^n - 2\left\{\sum_{j=1}^{2^n} x'_{k,j} \oplus F'_{\psi_{j-1}}(x_1,\ldots,x'_k,\ldots,x_n)\right\} \\ &\qquad 1 \leqslant k \leqslant n \end{aligned}\right\} \quad \ldots (2.12)$$

or , from equation (2.11) :

$$\left.\begin{aligned} R'_k &= 2^n - 2\left\{\sum_{j=1}^{2^n} x_{k,j} \oplus x_{l,j} \oplus F_{\ j-1}(x_1,\ldots,x_k,\ldots,x_n)\right\} \\ &\qquad 1 \leqslant k \leqslant n \end{aligned}\right\} \quad \ldots (2.13)$$

Now equation (2.13) is, by definition, equal to R_{kl} . That is $R'_k = R_{kl}$.

It can also be shown that $R'_{kl} = R_k$,

$R'_{klm} = R_{km}$, $R'_{km} = R_{klm}$

and $R'_1 = R_1$,

$R'_{1m} = R_{1m}$,

$R'_0 = R_0$ etc.

If this operation is extended to the replacement of x_k by x_{klm} the following results are obtained:

$R'_{klm} = R_k$, $R'_k = R_{klm}$,

$R'_{klmn} = R_{kn}$, $R'_{kn} = R_{klmn}$,

and $R'_{1m} = R_{1m}$,

$R'_0 = R_0$ etc.

It is important to note that this operation constitutes a re-ordering of the minterms of $F(x_1, \ldots, x_n)$ and that no information about the function is lost.

2.5 Spectral Translation.

Consideration of Operation 5 , above , gives rise to the following theorem :

2.5.1 The Theorem of Spectral Translation.

If in a Boolean function $F(x_1, \ldots, x_k, \ldots, x_n)$ having a spectrum $\langle R \rangle$, x_k is replaced by $\{x_a \oplus x_b \ldots \oplus x_h\} \oplus x_k$ where the set of subscripts $\langle a,b, \ldots ,h \rangle$ is denoted by $\langle S \rangle$, then the spectrum $\langle R' \rangle$ of the new function is generated from the spectrum $\langle R \rangle$ if :

in every subscript of the spectral coefficients of $\langle R \rangle$ containing k , the members of $\langle S \rangle$ are deleted if they exist, and appended if they do not.

— oOo —

Notes on the theorem

1/ When a first-order spectral coefficient is replaced by a higher-ordered coefficient under the above theorem , no other first-ordered spectral coefficient is replaced. This follows from the fact that no other first-order coefficient has the same subscript.

2/ If the operation of spectral translation is executed twice for the same variable replacement , the original spectrum results.

2.5.2 Interpretation and Implementation of Spectral Translation

Fig. 29a shows the implementation of the Boolean function $F(x_1, \ldots, x_k, \ldots, x_n)$ in terms of logic circuitry. Suppose that it is required to replace x_k by $x_k' = x_k \oplus x_1$. This is accomplished by means of an exclusive-OR gate and produces a new logic module $F'(x_1, \ldots, x_k', \ldots, x_n)$ as shown in Fig. 29b. The overall transfer function of the system remains unchanged ,from equation (2.11).

Fig. 29c shows the implementation of this operation for the variable x_k replaced by $x_k' = \{x_1 \oplus x_c \oplus x_f\} \oplus x_k$.

2.5.3 Significance of Spectral Translation.

2.5.3a In Logic Synthesis.

Because the theorem of spectral translation has the fundamental property of translating high-ordered spectral coefficients to low-ordered positions , it is clear that , in general , given a high-ordered function then a function of lower order may be generated from it. Now it has already been established that low-ordered functions have the property that they may be more easily synthesised in terms of threshold gates† and vertex (NAND,NOR,AND,OR) gates , than may high-ordered functions.

The fact that spectral translation itself is easily implemented by exclusive-OR gates means that a novel , and sometimes complete , synthesis procedure is possible, as will later be demonstrated.

---

† It is assumed that the exclusive-OR gate is an integral gate having a propagation delay comparable to that of a vertex gate.

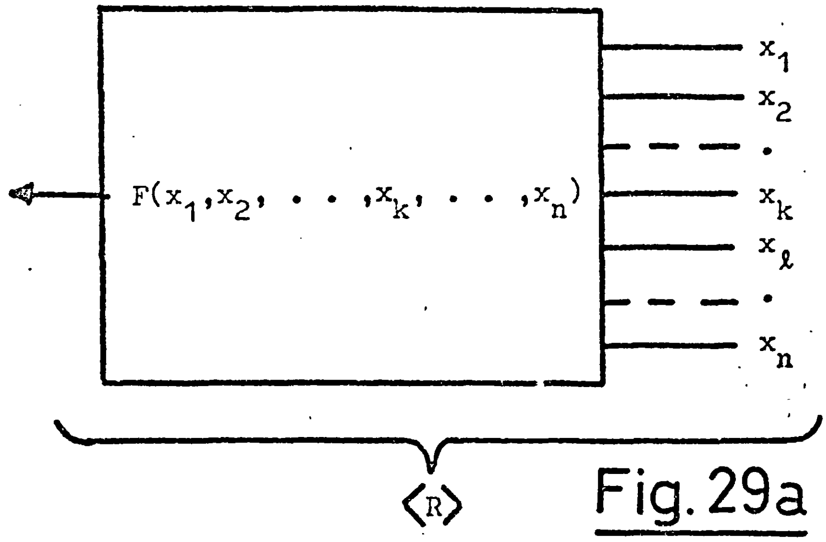


Fig. 29a

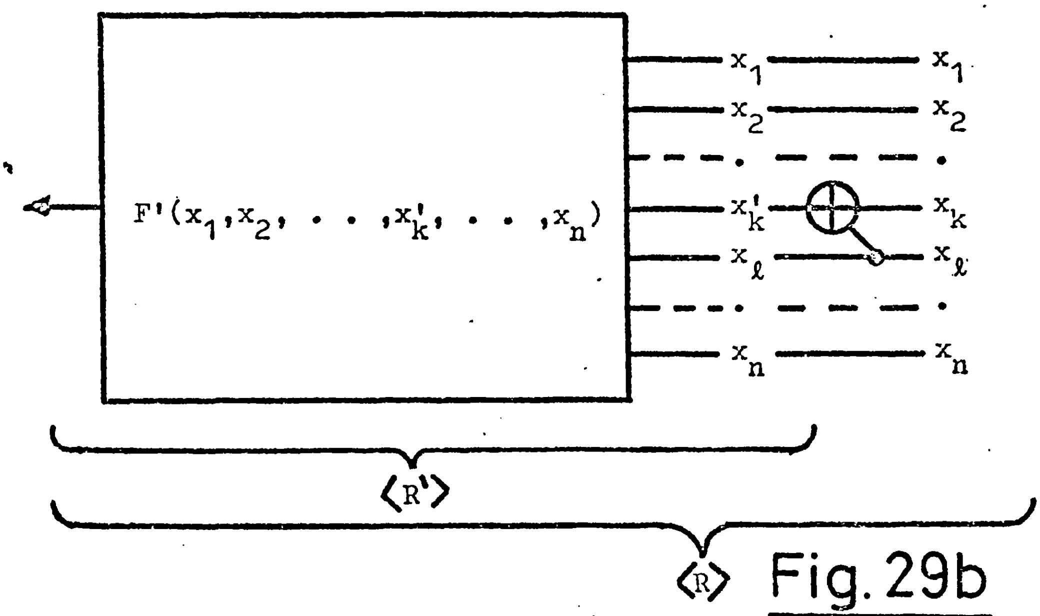


Fig. 29b

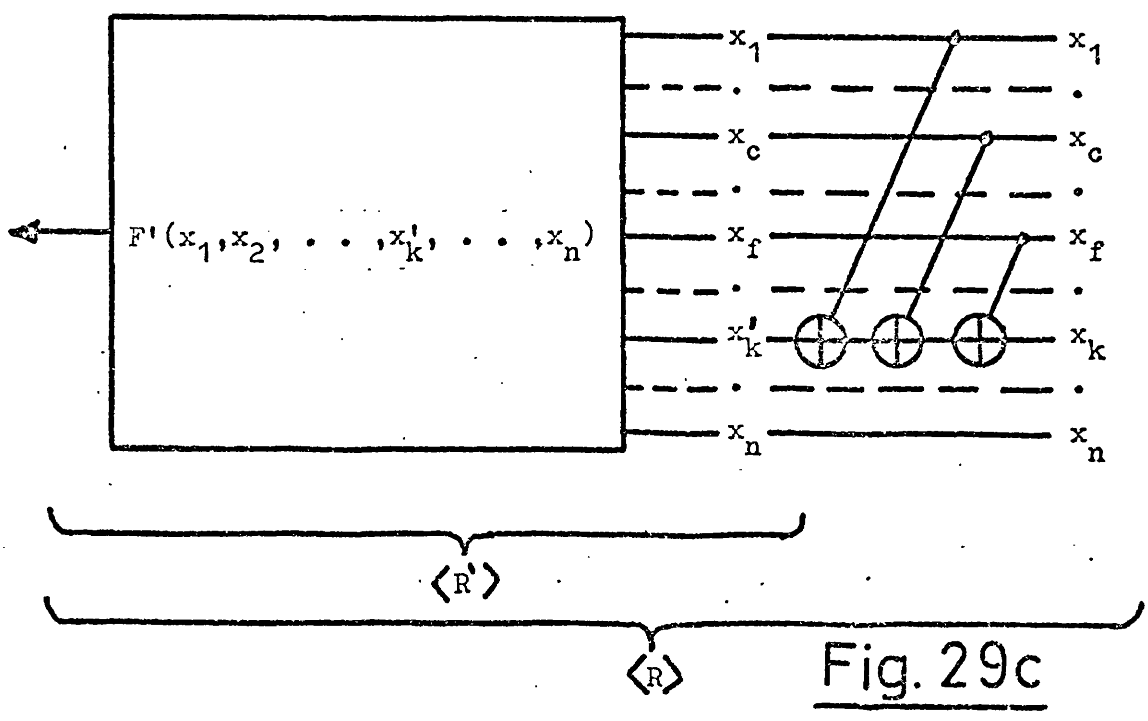


Fig. 29c

2.5.3b In Boolean Function Classification.

As has been mentioned above , Golomb, reference 26 , has shown that certain Boolean functions of given order n may be classified as 'equivalent' under the complementation and/or permutation of their defining variables.

In the light of the theorem of spectral translation a new , and more embracing , classification may be proposed.:

> A Boolean function $F_1(x_1, \ldots, x_n)$ of order n is classified as translationally-equivalent to another Boolean function $F_2(x_1, \ldots, x_n)$ of the same order,if $F_1(x_1, \ldots, x_n)$ can be mapped onto $F_2(x_1, \ldots, x_n)$ by the permutation and/or complementation of its defining variables and/or the , perhaps repeated , application of the theorem of spectral translation.

Clearly all Boolean functions which are equivalent fall into the sametranslationally-equivalent class. It follows that the number of translationally-equivalent functions which exist for a given n is smaller than the number of equivalent functions.

The practical importance of this new classification lies in the fact that translationally-equivalent functions can be synthesised from a representative , or canonic , function whose synthesis is known,by the complementation and/or permutation of the defining variables and/or the appending of suitable exclusive-OR logic.

If tables of representative canonic functions are generated , therefore , together with optimum syntheses , it is possible to synthesise any given function by

1/ establishing the translationally-equivalent class to which it belongs,

2/ finding the operations necessary to convert the given function to canonical form,

3/ to implement these operations in terms of logic circuitry, and then

4/ to append the optimum synthesis.

The choice of form of canonic function is arbitrary, but in order that an optimum synthesis be achieved it is clearly an advantage that the canonic function for each class should be predominantly first-ordered for reasons previously described.

With this in mind the following method of generating the canonic function in each class is proposed :

1/ Generate the lowest-ordered function possible in a given class by the operation of spectral translation.

2/ Render all first-order spectral coefficients positive (Operation 2).

3/ Permutate the defining variables so that the first-order spectral coefficients are arranged in descending order of magnitude, followed , where possible, by the second-order coefficients etc. (Operation 1).

This method has been used to generate a table of canonic functions for $n \leqslant 4$. This table appears in Appendix 3. The power of this form of Boolean function classification now becomes apparent. The total number of Boolean functions for $n \leqslant 4$ is 65,536 and under this classification the number of canonic functions is 18 . In practical terms this means that 18 unique logic modules are required to synthesise all possible Boolean functions, $n \leqslant 4$, under the application of the operations 1,2 and 5. Because this table does not specifically enumerate all possible complements of functions it is also necessary to invoke Operation 4.

Of these 18 functions one is trivial (function No.1) since it specifies a function with all false* minterms. It is worth noting at this stage that all but three of these canonic functions are threshold functions ,(the threshold functions are marked 'T'). The importance of this will become clear later.

The 'optimum syntheses' of these functions have not been shown since the definition of optimum will depend upon the criterion of optimality used. This may be minimum number of gates or inter-connections , cost etc.

It will be shown later that a more powerful classification method is possible but before embarking on the details of this it is necessary to investigate the application of spectral translation.

2.5.4 Application of Spectral Translation.

2.5.4a. Application to Synthesis by Threshold logic.

Dertouzos, reference 20 , has shown that a threshold function is uniquely characterised by the values of the first (n+1) spectral coefficients. These in fact are the Chow parameters , see reference 19. Moreover these coefficients may appear in any order and with any sign. All threshold functions are linearly separable and , because the evaluation of linearly separable functions is a complex procedure,tables of such functions have been prepared, see references 20 and 27. In these tables the first (n+1) spectral coefficients of each threshold function appear in ascending order of magnitude and are positive. These vectors are sufficient to characterise all n th order threshold functions and are called positive characteristic canonic vectors. In order to establish if a given function is a threshold function it suffices to arrange the first

* If operation 4 is invoked then this function characterises a function with all true minterms.

(n+1) spectral coefficients of the function in ascending order of magnitude, change all negative coefficients to positive and determine if this characteristic vector appears in the tables of of positive characteristic canonic vectors.

In order that the threshold gate corresponding to a particular canonic vector may be designed it is necessary to evaluate the weights associated with that vector. Again these threshold weights normally appear in the canonic vector tables. A representative set of such tables appears in Appendix 4.

The use of such tables is best illustrated by means of an example.

Consider the fourth-order function of Fig. 30.

The first (n+1) spectral coefficients of this function are

$$\begin{array}{ccccc} 4 & 12 & 4 & -4 & 0 \\ R_0 & R_1 & R_2 & R_3 & R_4 \end{array},$$

re-arranging these coefficients into ascending order of magnitude and changing all negative signs to positive the vector

12 4 4 4 0 is obtained.

Inspection of the tables of Appendix 4 , for n=4 , shows that this characteristic vector indeed defines a threshold function for which :

Characteristic vector	C	:	12	4	4	4	0
Weights	W	:	2	1	1	1	0

Now because there is a one-to-one correspondence between each weight and associated member of the characteristic vector , both in magnitude and sign , it is possible to re-express the original function in terms of the weights by re-arrangement and change of sign as appropriate.

In this example

	4	12	4	-4	0	are the original coefficients
	R_0	R_1	R_2	R_3	R_4	
and	1	2	1	-1	0	are the corresponding weights.
	w'_0	w'_1	w'_2	w'_3	w'_4	

From these weights the parameters of the threshold gate may be calculated. For a more detailed treatment see references 20 and 28.

The input weightings for each gate input are given by :

Weighting at input x_i is equal to w'_i , $1 \leqslant i \leqslant n$ . . . (2.14)

The output weighting of the gate is given by :

$$\text{Weighting at output}^{\dagger} = \tfrac{1}{2}\left\{\left(\sum_{i=1}^{n} |w'_i|\right) + w'_0 + 1\right\} \qquad \ldots (2.15)$$

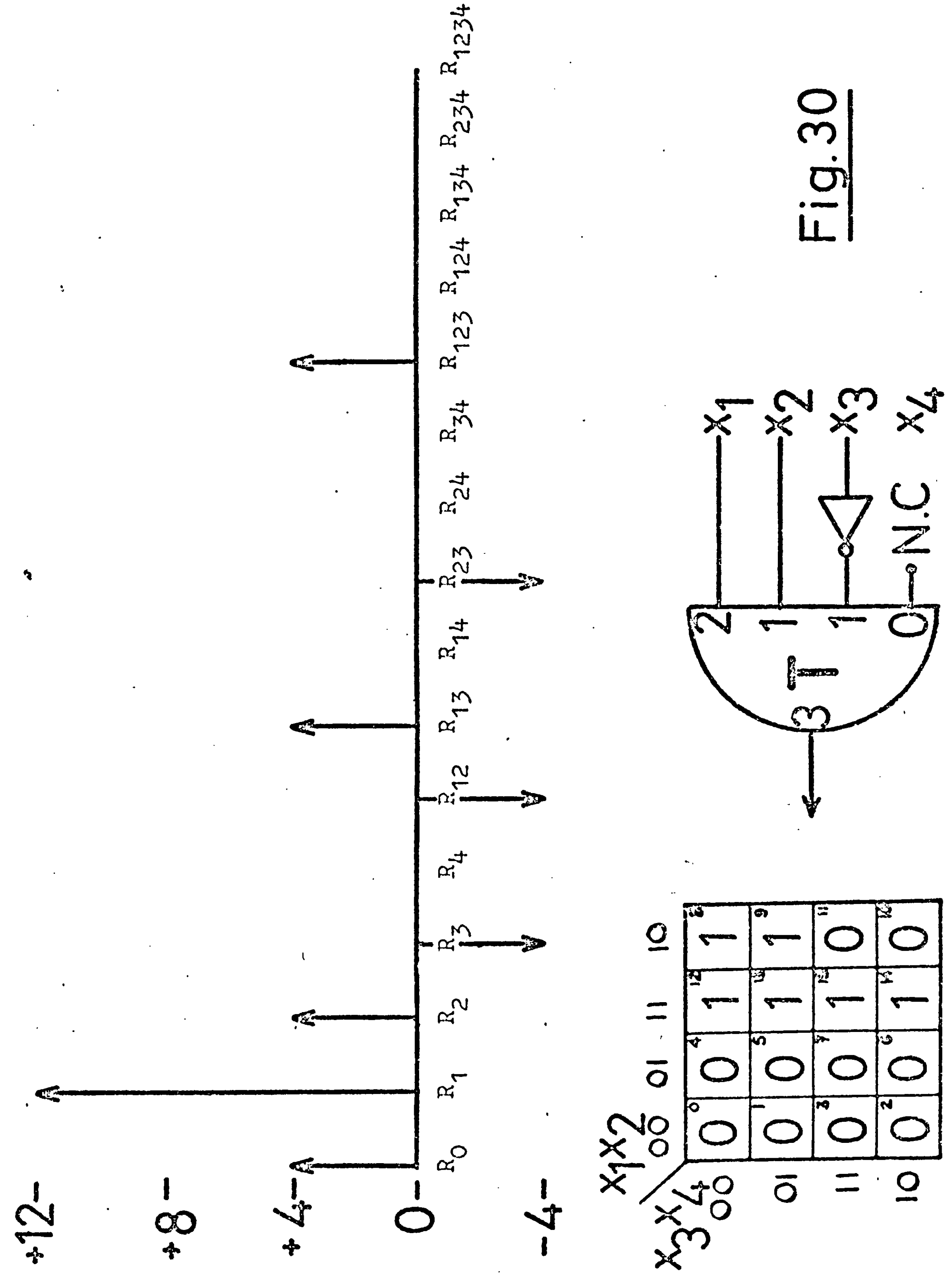
As threshold gates with a negative weight capability will not be considered it is important to note that if any w'_i are negative the respective input must be complemented and the corresponding weight changed in sign. In this particular example therefore, w'_3 is changed in sign and an inverter is placed before input x_3.

From equation (2.15), the weighting at the output of this gate is $\tfrac{1}{2}(4 + 1 + 1) = 3$. The gate is shown in Fig. 30.

Note that the input weighting of 0 is equivalent to a no-connection. That is , the original function is independent of variable x_4. (The function is in fact third-ordered).

The description of the operation of this gate is now straightforward. Clearly if x_1 and x_2 have the value 1 then the output

$\dagger$ Note that some authors define this weighting with $-w'_0$, this is because Chow parameters were not originally defined using the Rademacher/Walsh transform . This results in a difference of sign for R_0.



+12
+8
+4
0
-4
R_0 R_1 R_2 R_3 R_4 R_{12} R_{13} R_{14} R_{23} R_{24} R_{34} R_{123} R_{124} R_{134} R_{234} R_{1234}
x_1x_2
x_3x_4
00 01 11 10
x_1 x_2 x_3 x_4
2 1 1 0
T
3
N.C

Fig.30

threshold of 3 will be equalled since x_1 is weighted 2 and x_2 is weighted 1 . The gate will thus give an output of one. Similarly the gate will also give an output of one if $x_1 = 1$ and $x_3 = 0$ since x_3 is complemented. Also if $x_1 = 1$, $x_2 = 1$ and $x_3 = 0$ the sum of the weights at the input is 4 which exceeds the output threshold 3 , the gate output will then again be 1 . In all other cases the output threshold is not reached so that the gate output is 0 .

The gate function may therefore be concluded to be

$x_1.(x_2 + \bar{x}_3)$, where '.' signifies logical AND ,
'+' signifies logical OR.

This result can be checked from the Karnaugh map of the function shown in Fig. 30 .

The role of the spectral translation operation in the synthesis of Boolean functions by means of threshold functions is now considered by means of a simple example.

Given : the function shown on the Karnaugh map of Fig. 31.

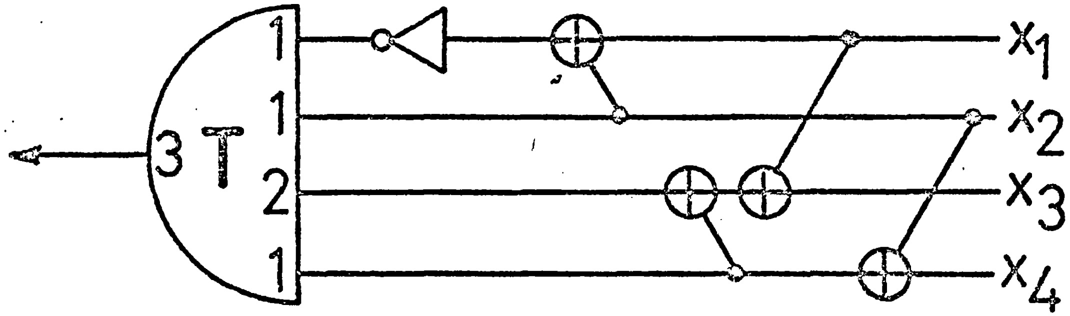
The spectrum of this function is as follows

0	0	4	0	0	-4	0	0
R_0	R_1	R_2	R_3	R_4	R_{12}	R_{13}	R_{14}
4	4	0	-4	-4	0	4	12
R_{23}	R_{24}	R_{34}	R_{123}	R_{124}	R_{134}	R_{234}	R_{1234}

If the first (n+1) spectral coefficients of this function are ordered by magnitude and rendered positive the result is :

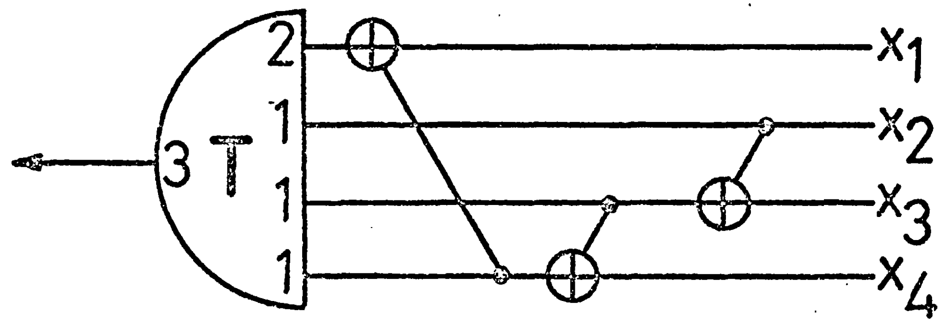
4 0 0 0 0 which does not appear in the tables of positive characteristic vectors (Appendix 4) , that is , it is not a threshold function.

Now apply the operation of spectral translation to generate a new spectrum $\langle R' \rangle$ from the above spectrum $\langle R \rangle$, where $R_4' = R_{24}$:



x_3x_4 \ x_1x_2	00	01	11	10
00	0	1	1	0
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

First Solution. Fig. 31a



Optimised Solution Fig. 31b (See p.132)

0	0	4	0	4	-4	0	-4
R'_0	R'_1	R'_2	R'_3	R'_4	R'_{12}	R'_{13}	R'_{14}

4	0	4	-4	0	12	0	0
R'_{23}	R'_{24}	R'_{34}	R'_{123}	R'_{124}	R'_{134}	R'_{234}	R'_{1234}

Applying the operation again for the generation of a new spectrum $\langle R'' \rangle$ from $\langle R' \rangle$, where $R''_3 = R'_{134}$:

0	0	4	12	4	-4	4	-4
R''_0	R''_1	R''_2	R''_3	R''_4	R''_{12}	R''_{13}	R''_{14}

0	0	0	0	0	0	-4	4
R''_{23}	R''_{24}	R''_{34}	R''_{123}	R''_{124}	R''_{134}	R''_{234}	R''_{1234}

Finally , applying the operation for the generation of a new spectrum $\langle R''' \rangle$ from $\langle R'' \rangle$, where $R'''_1 = R''_{12}$:

0	-4	4	12	4	0	0	0
R'''_0	R'''_1	R'''_2	R'''_3	R'''_4	R'''_{12}	R'''_{13}	R'''_{14}

0	0	0	4	-4	4	-4	0
R'''_{23}	R'''_{24}	R'''_{34}	R'''_{123}	R'''_{124}	R'''_{134}	R'''_{234}	R'''_{1234}

Now if the first (n+1) spectral coefficients of this function are ordered by magnitude and rendered positive the result is :

12 4 4 4 0 which appears in the tables of positive characteristic vectors (Appendix 4) , that is , it is a threshold function.

The threshold gate parameters may now be calculated using the method described above :

The coefficients	0	-4	4	12	4	give the
	R'''_0	R'''_1	R'''_2	R'''_3	R'''_4	
corresponding weights	0	-1	1	2	1	, see Appendix 4 .
	w'_0	w'_1	w'_2	w'_3	w'_4	

From equation (2.15) , the output weight is

$$\tfrac{1}{2}(5 + 0 + 1) = 3 .$$

The resulting gate appears in Fig. 31a together with the exclusive-OR circuitry necessary to carry out the spectral translations.

That is, initially x_4 is replaced by $x_2 \oplus x_4$ and so on .

Because w_1' is negative an inverter is placed on the input line x_1 before the gate.

This example illustrates a property common to many non-threshold Boolean functions , that is that such functions may be rendered linerarly-separable (threshold functions) , by the application of the operation of spectral translation. Such functions will be said to have threshold functions 'embedded' within them.

The importance of this result of course lies in the fact that the versatility of threshold logic is increased many-fold by the straightforward appending of equivalence (exclusive-OR) -type logic.

In fact the tables of Appendix 3 show that there are only three classes of functions out of eighteen which do not have embedded threshold functions , $n \leqslant 4$.

It has been argued*that the continuing non-appearance of any satisfactory technology for making threshold gates commercially available limits the practical usefulness of these methods. In fact the difficulties in the fabrication of these gates have been overcome by a novel design method devised by Dr.S.L. Hurst , University of Bath. The implications of the use of this gate are discussed in Chapter 3 .

In practice the application of spectral translation to convert a high-ordered function into a low-ordered function,so that embedded threshold functions may be employed in the synthesis of given functions, may be carried out in several different ways. Each of the alternative methods for carrying out the translations results in a differing number of gates employed in the final

---

* Referee's comment on paper on this subject submitted to I.E.E.E. Transactions on Computers by the author.

synthesis. The criteria governing the optimum choice of spectral translations for the minimisation of the number of gates used in a given synthesis appears in Section 2.5.5.

2.5.4b Application to Synthesis by Vertex logic.

As explained previously , functions having high-ordered spectra are generally more difficult to synthesise using vertex (AND,OR,NAND,NOR) logic than are functions with low-ordered spectra because their true minterms do not fall predominantly into areas corresponding to the intersection or union of any particular defining variables.

It has been shown however that the application of the operation of spectral translation enables a high-ordered function to be re-expressed as a function of lower order under exclusive-OR synthesis.

The techniques of spectral translation can therefore be used , without the necessity of employing threshold gates , to problems employing conventional vertex gates. Moreover the synthesis of Boolean functions by this method gives rise , in general , to more elegant solutions than would be the case in circuits employing no exclusive-OR gates. This follows from the observation that exclusive-OR functions are not easily synthesised by vertex logic.

Consider the function given by the Karnaugh map of Fig. 32a. This function has the spectrum

2	2	2	2	6	2	-6	6
R_0	R_1	R_2	R_3	R_4	R_{12}	R_{13}	R_{14}
-6	-2	6	2	-2	-2	6	-2
R_{23}	R_{24}	R_{34}	R_{123}	R_{124}	R_{134}	R_{234}	R_{1234}

Note : this function does not have an embedded threshold function.

Applying the operation of spectral translation to generate a new spectrum $\langle R' \rangle$ from the above spectrum $\langle R \rangle$,where $R_1' = R_{14}$:

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	0	1	1	0
10	0	0	1	1

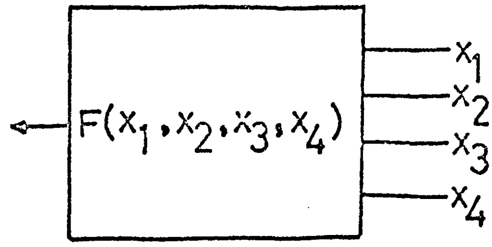


Fig. 32a. Sample function.

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	0	1	1	0
10	0	0	1	1

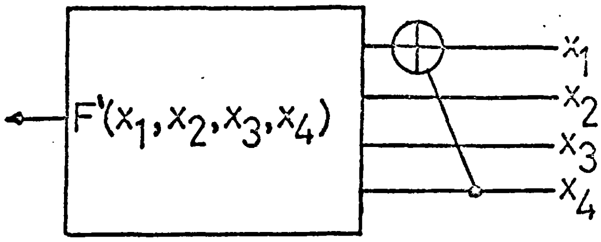


Fig 32b. Function after 1st. translation.

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	1	0	1	1
10	0	0	1	1

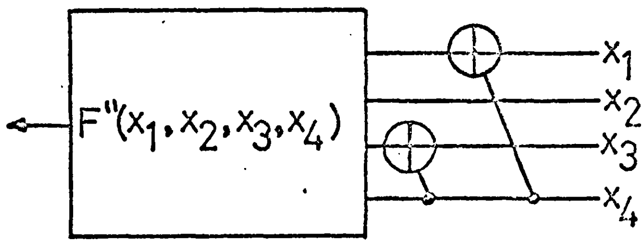


Fig. 32c. Function after 2nd. translation.

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	1
10	0	0	1	1

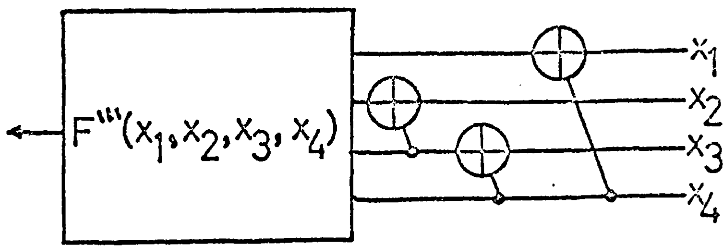


Fig. 32d. Function after 3rd. translation.

$$\begin{array}{cccccccc} 2 & 6 & 2 & 2 & 6 & -2 & -2 & 2 \\ R'_0 & R'_1 & R'_2 & R'_3 & R'_4 & R'_{12} & R'_{13} & R'_{14} \\ \\ -6 & -2 & 6 & -2 & 2 & -6 & 6 & 2 \\ R'_{23} & R'_{24} & R'_{34} & R'_{123} & R'_{124} & R'_{134} & R'_{234} & R'_{1234} \end{array}$$

Again , generating a new spectrum $\langle R'' \rangle$ from the above spectrum $\langle R \rangle$ where $R''_3 = R'_{34}$:

$$\begin{array}{cccccccc} 2 & 6 & 2 & 6 & 6 & -2 & -6 & 2 \\ R''_0 & R''_1 & R''_2 & R''_3 & R''_4 & R''_{12} & R''_{13} & R''_{14} \\ \\ 6 & -2 & 2 & 2 & 2 & -2 & -6 & -2 \\ R''_{23} & R''_{24} & R''_{34} & R''_{123} & R''_{124} & R''_{134} & R''_{234} & R''_{1234} \end{array}$$

Finally , generating a new spectrum $\langle R''' \rangle$ from the above spectrum $\langle R'' \rangle$, where $R'''_2 = R''_{23}$:

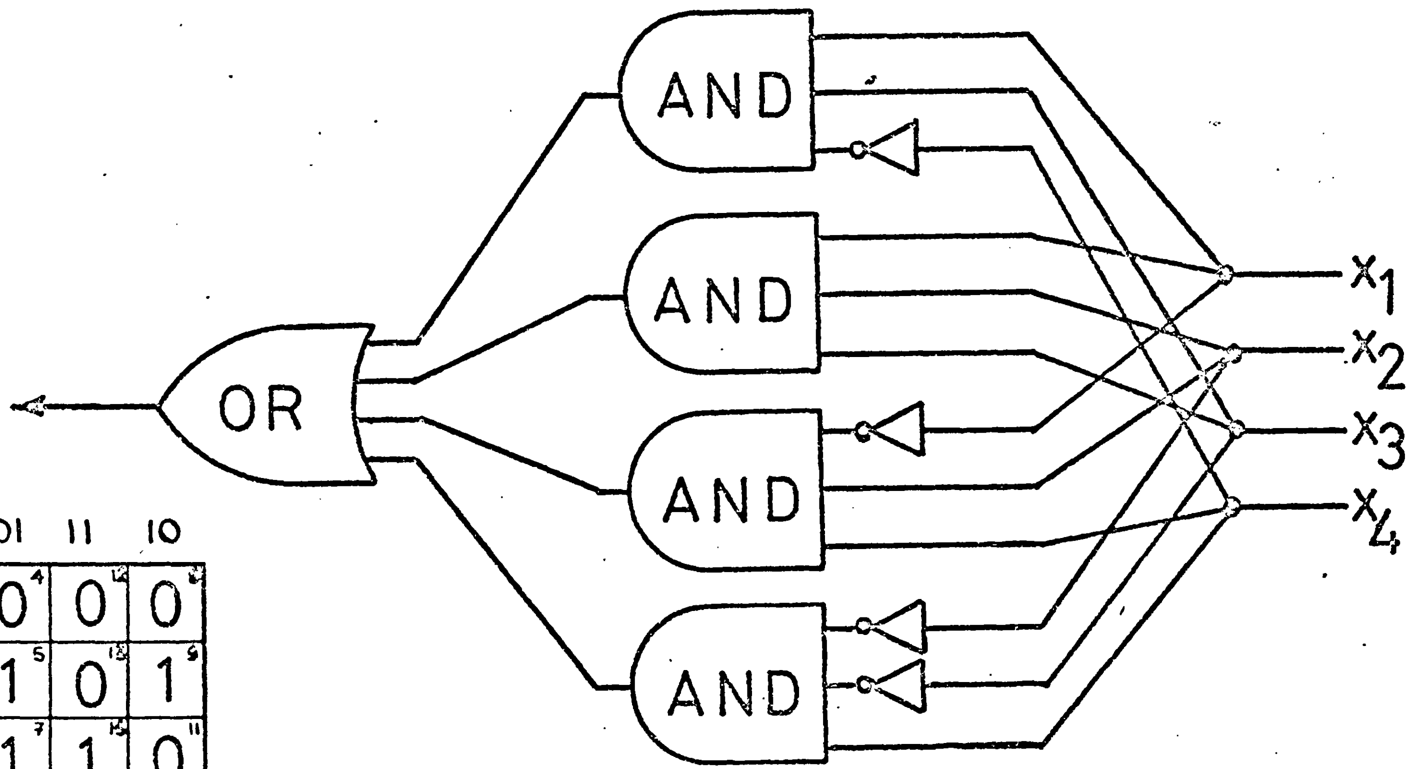
$$\begin{array}{cccccccc} 2 & 6 & 6 & 6 & 6 & 2 & -6 & 2 \\ R'''_0 & R'''_1 & R'''_2 & R'''_3 & R'''_4 & R'''_{12} & R'''_{13} & R'''_{14} \\ \\ 2 & -6 & 2 & -2 & -2 & -2 & -2 & 2 \\ R'''_{23} & R'''_{24} & R'''_{34} & R'''_{123} & R'''_{124} & R'''_{134} & R'''_{234} & R'''_{1234} \end{array}$$

A point has now been reached where the spectrum is maximally first-ordered , that is to say no further translations can increase the magnitudes of the first (n+1) coefficients.

The functions generated by each of these translations are shown in Figs. 32b , 32c and 32d respectively. Note that at each step the true minterms of the function tend to come together in larger groups; that is , the true minterms fall more predominantly in areas corresponding to the intersection of the defining variables.

Fig.33a shows a simple , conventional two-level synthesis (AND,OR) of the original function of Fig. 32a together with necessary inverters. The same figure shows the synthesis accomplished with the aid of the above translations , implemented by exclusive-OR gates, Fig. 33b.

The saving in circuit complexity is considerable in this



x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	0	1	1	0
10	0	0	1	1

Fig.33a

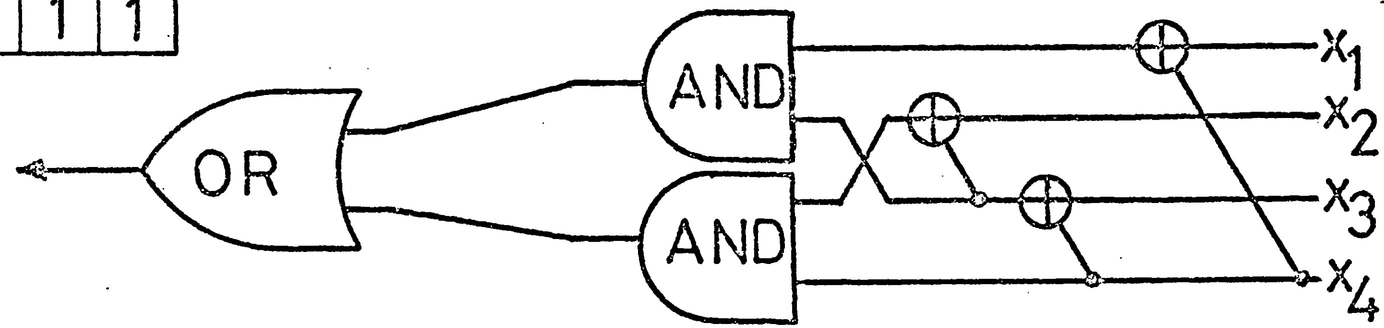


Fig.33b

example , the number of interconnections required being 20 and 12 respectively.

It is worth noting that because only positive spectral coefficients have been translated no inverters are required in the latter synthesis. This would not necessarily be the case , of course , if NAND,NOR logic were employed.

In the case of threshold logic synthesis it was noted that the appearance of a 0 in the weighting vector w_i' , $1 \leq i \leq n$, implied a no-connection , that is the function was independent of variable x_i . It is true of all functions that if 0 appears in every spectral coefficient having a subscript containing i then that function is independent of x_i. It is clear by inspection that the function considered here has no variable redundancies.

Again the spectral translations in this example have been carried out with no obvious plan to minimise the number of gates generated. In fact this solution does employ the minimum number of necessary exclusive-OR gates for reasons developed in the next section.

2.5.5 Gate Minimisation Criteria.

In order that the minimisation criteria pertaining to the synthesis of digital circuits under the operation of spectral translation may be developed it is necessary to employ Galois Field 2 theory. For this reason reference should be made to Appendix 2 before proceeding with this section.

A GF(2) matrix is able to represent an operation of the type : replace x_i by $x_i' = x_i \oplus x_j$,which corresponds to a spectral translation. For example : replace x_1 by $x_1' = x_1 \oplus x_2$ would be represented as

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} (x_1 \oplus x_2) \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad , \text{ in GF(2) .}$$

In field theory a matrix of this type , where the main diagonal consists of allowed values other than zero and only one other allowed value , other than zero , appears off the main diagonal , defines an elementary operation . An elementary operation thus corresponds to a spectral translation where a second-order spectral coefficient replaces a first-order spectral coefficient, since if x_i is replaced by $x_i' = x_i \oplus x_j$ then R_i is replaced by $R_i' = R_{ij}$.

It also follows that if it is required to represent a spectral translation where a spectral coefficient of above second-order replaces a first-order coefficient then this can be achieved by the multiplication of a number of suitable matrices in GF(2) , each of which define an elementary operation of the type above.

For example , the replacement of x_1 by $x_1' = x_1 \oplus x_2 \oplus x_3$ can be represented by

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} (x_1 \oplus x_2 \oplus x_3) \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2 \\ x_3 \\ x_4 \end{bmatrix},$$

where x_1 has first been replaced by $x^* = x_1 \oplus x_2$ and then x_1 has been replaced by $x_1' = x_1^* \oplus x_3 = x_1 \oplus x_2 \oplus x_3$.

In general, a series of elementary operations in GF(2) can represent any single spectral translation.

These ideas may be extended to the representation of several consecutive spectral translations. For instance, in the example of the previous section the overall result of the series of spectral translations was to replace x_1 by $x_1''' = x_1 \oplus x_4$, x_2 by $x_2''' = x_2 \oplus x_3 \oplus x_4$,

x_3 by $x_3''' = x_3 \oplus x_4$ and x_4 by $x_4''' = x_4$. See also Fig. 33.

The result of this series of translations can thus be represented as

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} (x_1 \oplus x_4) \\ (x_2 \oplus x_3 \oplus x_4) \\ (x_3 \oplus x_4) \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1''' \\ x_2''' \\ x_3''' \\ x_4''' \end{bmatrix} \quad \text{, in GF(2) .}$$

It follows that the above matrix may be re-expressed in terms of a number of matrices , in GF(2) , each representing an elementary operation which corresponds to the spectral translation of a second-order spectral coefficient to a first order position.

Now it is a property of GF(2), and indeed any field*, that the matrix resulting from the multiplication of a series of matrices, each matrix defining an elementary operation , has a determinant which is non-zero. (In the case of GF(2) the matrix has a determinant of value 1).

It is therefore possible to test the validity of a proposed series of spectral translations in the following way :

<u>Test 1.</u>

If the result of a proposed series of spectral translations is represented as a matrix $[\Lambda]$ in GF(2) , then such a series of translations is possible only if the determinant of $[\Lambda]$ has the value 1 .

—oOo—

eg. for the last example

$$[\Lambda] = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

* The author is indebted to Mr. B.Ireland ,University of Bath , for his advice on the aspects of field theory discussed here.

Expanding the determinant of $[\Lambda]$ by the first column in the usual way gives

$$|\Lambda| = 1.\begin{vmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix} = 1.1.\begin{vmatrix} 1 & 1 \\ 0 & 1 \end{vmatrix}$$

$$= 1.1.\left\{(1.1) + (0.1)\right\}$$

$$= 1.1.(1 + 0)$$

$$= 1.1.1$$

$$= 1$$

where '.' and '+' denote multiplication and addition in GF(2) respectively. See Appendix 2 .

This result shows that a series of spectral translations is possible for this example.

One other elementary operation exists in GF(2) which can be shown to correspond to the interchange of defining variables. (Operation 1 , section 2.4). This is equivalent to an interchange of the rows of $[\Lambda]$ which does not invalidate Test 1 and is implemented by a simple interchange if input lines to the final logic module of the circuit.

The functions defined by $[\Lambda]$, where $|\Lambda| = 1$, are called a <u>Basis</u> and spectral translation is equivalent to a <u>Basis Transformation</u>.

Note that Test 1 is sufficient to define a basis but does not give any information about the spectral translations , and thus number of gates , necessary to generate that basis. Test 1 then does not assist in the gate minimisation problem.

It has been shown that spectral translation is best used , from a synthesis point of view , in mapping a high-ordered function onto a lower-ordered function. The most significant spectral coefficients are then translated to first-ordered positions. It follows that the choice of basis is made from the set of spectral coefficients whose magnitudes are the greatest.

For the example of Fig. 33 the spectrum is

2	2	2	2	6	2	-6	6
R_0	R_1	R_2	R_3	R_4	R_{12}	R_{13}	R_{14}
-6	-2	6	2	-2	-2	6	-2
R_{23}	R_{24}	R_{34}	R_{123}	R_{124}	R_{134}	R_{234}	R_{1234}

The most significant spectral coefficients are $R_4, R_{13}, R_{14}, R_{23}$, R_{34} and R_{234} , each of which have a magnitude of 6. The basis is therefore chosen from the functions x_4, $x_1 \oplus x_3$, $x_1 \oplus x_4$, $x_2 \oplus x_3$, $x_3 \oplus x_4$ and $x_2 \oplus x_3 \oplus x_4$. Of course if no set of these functions form a basis it would be necessary to include other functions whose corresponding spectral coefficients have a magnitude of 2 .

Once a basis has been chosen , that is a set of n of such functions satisfying Test 1 , it is required to find the minimum number of exclusive-OR gates which will generate that basis. A method which enables such a basis to be generated using the minimum number of exclusive-OR gates is given, by means of an example, below.

Suppose , for the function of Fig. 33 the following set of functions is chosen :

Function No.	Function
1	$x_1 \oplus x_4$
2	$x_2 \oplus x_3 \oplus x_4$
3	$x_3 \oplus x_4$
4	x_4

The corresponding $[\Lambda]$ matrix is $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

It has already been established , see above , that this matrix has a determinant of value 1 , and therefore passes Test 1 . These functions therefore form a basis.

Now inspection of these functions shows that function 4 can be generated without employing any exclusive-OR gates , function 3 can be generated using one exclusive-OR gate, function 2 requires two exclusive-OR gates and function 1 requires one exclusive-OR gate. In addition , function 3 can be generated from function 4 using one exclusive-OR gate since $x_3 \oplus x_4 = x_3 \oplus \{x_4\}$, function 2 can be generated from function 3 using one exclusive-OR gate since $x_2 \oplus x_3 \oplus x_4 = x_2 \oplus \{x_3 \oplus x_4\}$ and function 1 can be generated from function 2 using three exclusive-OR gates since $x_1 \oplus x_4 = x_1 \oplus x_2 \oplus x_3 \oplus \{x_2 \oplus x_3 \oplus x_4\}$ etc. These results can be obtained directly from the $[\Lambda]$ matrix by noting that

1/ The number of exclusive-OR gates required to synthesise any basis function is given by $\{$the number of 1's appearing in the corresponding row of $[\Lambda]\}$ - 1 .

2/ The number of exclusive-OR gates required to generate the i th basis function from the j th basis function is given by the number of differences between the i th and j th rows of $[\Lambda]$.

This information is best presented as a difference table , denoted as Δ . For the above basis the Δ table is

		Function No.			
		1	2	3	4
Fn. No.	1	1	3	2	1
	2	3	2	1	2
	3	2	1	1	1
	4	1	2	1	0

, where the entries $\delta_{i,i}$, $1 \leqslant i \leqslant n$, are the number of exclusive-OR gates required to synthesis the i th basis function and the entries $\delta_{i,j}$, $1 \leqslant i,j \leqslant n$, are the number of exclusive-OR gates required to generate the i th basis function from the j th basis function. From the result $x_i \oplus x_j = x_j \oplus x_i$ it follows that $\delta_{i,j} = \delta_{j,i}$. Because of this symmetry only a part of this table need be generated. In this example

	1	2	3	4
1	1			
2	3	2		
3	2	1	1	
4	1	2	1	0

contains all the required information.

Suppose that it is first decided to generate the fourth function of the basis. This is an obvious choice because no gates are required. $\delta_{4,4}$ is then ringed and the fourth row and column of Δ are ticked to show that they are available for the generation of the remaining functions.

	1	2	3	✓ 4
1	1			
2	3	2		
3	2	1	1	
✓ 4	1	2	1	0

. Now several equally attractive alternatives are possible. Functions 1 or 3 may be generated from function 4 using only one gate. On the other hand functions 1 or 3 may be generated directly using only one gate. Suppose that in this case it is decided to generate functions 1 and 3 directly , the Δ table then becomes

	✓ 1	2	✓ 3	✓ 4
✓ 1	1			
2	3	2		
✓ 3	2	1	1	
✓ 4	1	2	1	0

. Now only function 2 remains to be synthesised. The minimum number of gates necessary to do this is one if function 2 is generated from function 3 , which is available. This gives the final Δ table as

	✓ 1	✓ 2	✓ 3	✓ 4
✓ 1	1			
✓ 2	3	2		
✓ 3	2	1	1	
✓ 4	1	2	1	0

. All the basis functions have now been

synthesised and the total number of gates used, which is the sum of the ringed numbers, is three* In practice, when an equal choice is presented between elements on the diagonal of $\triangle$ and elements not on the diagonal , the diagonal elements are chosen . This reduces the propagation time of the final circuit.

In general , if a basis is chosen where a spectral translation from say, third order to first order is implied then it is clear that at least two exclusive-OR gates will be required , irrespective of the actual method of synthesis. This observation gives rise to

Lemma 1.

The absolute minimum number of exclusive-OR gates required to synthesise a basis is equal to the highest number of exclusive-OR gates required to generate any function of that basis.

—oOo—

In the example above the basis function requiring the highest number of exclusive-OR gates for its direct generation is function 2 which requires two gates . The absolute minimum number of gates required to synthesise the basis is thus two, which is one gate less than that found necessary in practice.

The minimisation of the number of exclusive-OR gates required to convert a function to its maximally first-ordered form is given by :

1/ Arrange the spectral coefficients in order of magnitude. (Excepting R_0)

2/ Find the bases which correspond to the highest and equal-highest magnitude sets of spectral coefficients.

3/ Apply the gate minimisation procedure to each of these candidate bases in turn.

4/ Select the solution giving the minimum number of gates.

In practice the number of candidate bases , $n \leqslant 7$, turns

* Fig. 33 shows the implementation of this solution. (See p.123)

out to be small. This procedure is therefore quickly executed by means of the digital computer.

In the case of threshold logic, where a negative weight capability does not exist, it has been shown that for every negative valued spectral coefficient translated to first-order a complementing gate must be introduced in the final circuit. If therefore it is required to minimise the number of gates under these circumstances a modified minimisation procedure must be employed.

As an illustration of these methods consider the function shown in Fig. 31, p 117. The circuit of Fig. 31a was synthesised without regard to gate minimisation by the repeated application of spectral translation. See Section 2.5.4a. If gate minimisation is employed however the circuit of Fig. 31b results, which shows both a saving of one exclusive-OR gate and one inverter gate together with a reduction in circuit complexity.

2.6 Disjoint Spectral Translation.

2.6.1 Defining operation.

An operation* will now be considered which differs in implementation from those considered above in that a feed-forward signal path is created.

Operation 6

The interchange of spectral coefficients R_0 and R_k, $1 \leqslant k \leqslant n$.

Let the given function be $F(x_1,x_2, \ldots ,x_k, \ldots ,x_n)$.

Define a new function by $F(x_1,x_2, \ldots ,x_k, \ldots ,x_n)$

$$\triangleq x_k \oplus F'(x_1,x_2, \ldots ,x_k, \ldots ,x_n) \quad \ldots (2.16)$$

---

* Dertouzos has considered an operation similar to this under the heading of 'equidualisation', Ref. 20.

where the given function has the spectrum $\langle R \rangle$ and the new function $F'(x_1,x_2, \ldots ,x_k, \ldots ,x_n)$ has the spectrum $\langle R' \rangle$.

Substitution of equation (2.7) in equation (2.16) gives

$$R_k = 2^n - 2\left\{\sum_{j=1}^{2^n} x_{k,j} \oplus F_{\psi_{j-1}}(x_1,x_2, \ldots ,x_k, \ldots ,x_n)\right\}$$

$$= 2^n - 2\left\{\sum_{j=1}^{2^n} x_{k,j} \oplus x_{k,j} \oplus F'_{\psi_{j-1}}(x_1,x_2, \ldots ,x_k, \ldots ,x_n)\right\} \quad \ldots (2.17)$$

$$1 \leqslant k \leqslant n \quad .$$

But $x_{k,j} \oplus x_{k,j} \triangleq 0 \triangleq x_{0,j}$,see section 2.2.2, and the right hand side of equation (2.17) reduces to

$$2^n - 2\left\{\sum_{j=1}^{2^n} x_{0,j} \oplus F'_{\psi_{j-1}}(x_1,x_2, \ldots ,x_k, \ldots ,x_n)\right\} \quad \ldots (2.18)$$

which is by definition equal to R'_0 . See section 2.2.2 .

Similarly

$$R_1 = 2^n - 2\left\{\sum_{j=1}^{2^n} x_{1,j} \oplus F_{\psi_{j-1}}(x_1,x_2, \ldots ,x_k, \ldots ,x_n)\right\}$$

$$= 2^n - 2\left\{\sum_{j=1}^{2^n} x_{1,j} \oplus x_{k,j} \oplus F'_{\psi_{j-1}}(x_1,x_2, \ldots ,x_k, \ldots ,x_n)\right\}. \quad (2.19)$$

$$\triangleq R'_{k1}$$

It can also be shown that

$$R'_k = R'_0 \quad ,$$

$$R'_1 = R_{k1} \quad ,$$

$$R'_{k1m} = R_{1m} \quad , \quad R'_{1m} = R_{k1m} \quad \text{etc.}$$

These results give rise to the following theorem :

2.6.2 <u>The Theorem of Disjoint Spectral Translation</u>.

If , given a Boolean function $F(x_1, \ldots ,x_k, \ldots ,x_n)$ having a spectrum $\langle R \rangle$ it is required to generate a new function

$F'(x_1, \ldots, x_k, \ldots, x_n)$ having a spectrum R', where $F(x_1, \ldots, x_k, \ldots, x_n) \triangleq x_k \oplus F'(x_1, \ldots, x_k, \ldots, x_n)$ then $\langle R' \rangle$ may be generated from $\langle R \rangle$ if :

in every subscript of the spectral coefficients of $\langle R \rangle$ k is deleted if it exists and is appended if it does not.

—oOo—

Notes on the theorem .

1/ The theorem is termed 'disjoint' because it enables one of the defining variables of the original function to be separated from its fellows and gives rise to a feed-forward signal path , as described below. Unlike the operations that have so far been considered , disjoint spectral translation has the property that it can , where applicable , convert one function to another even though the functions have different ratios of true/false minterms.

2/ For the special spectral coefficients R_0, R_k the theorem is applied as follows :

$$R_0 \triangleq R'_{0k} \triangleq R'_k \quad ,$$
$$R_k \triangleq R'_{\cancel{k}} \triangleq R'_0 \quad .$$

3/ The theorem defines an operation which allows the zero-ordered spectral coefficient of any Boolean function to be interchanged with any first-ordered spectral coefficient. If the operation is repeated it follows that the zero-ordered coefficient may be interchanged with *any* spectral coefficient.

2.6.3 Interpretation and Implementation of Disjoint Spectral Translation.

Fig. 34a shows the implementation of the Boolean function $F(x_1, \ldots, x_k, \ldots, x_n)$ having a spectrum $\langle R \rangle$. According to the above this function is replaced by $x_k \oplus F'(x_1, \ldots, x_k, \ldots, x_n)$ where $F'(x_1, \ldots, x_k, \ldots, x_n)$ is a new function with spectrum $\langle R' \rangle$. This implementation is shown in Fig. 34b. The overall transfer

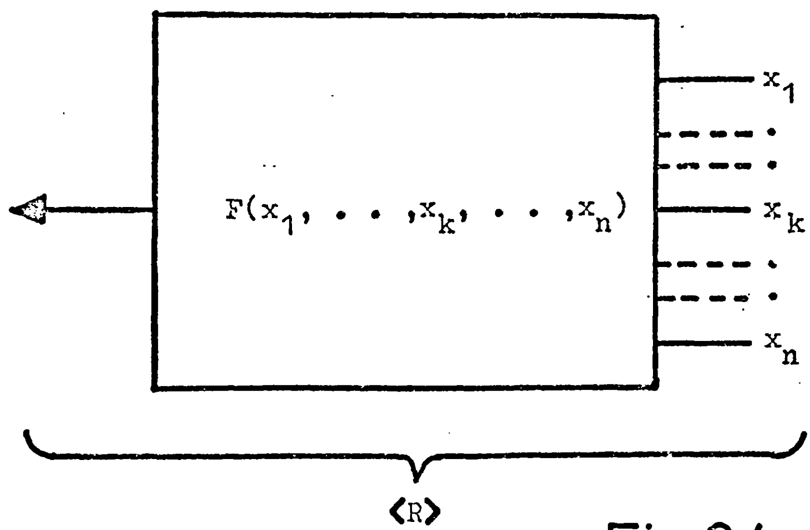


Fig.34a

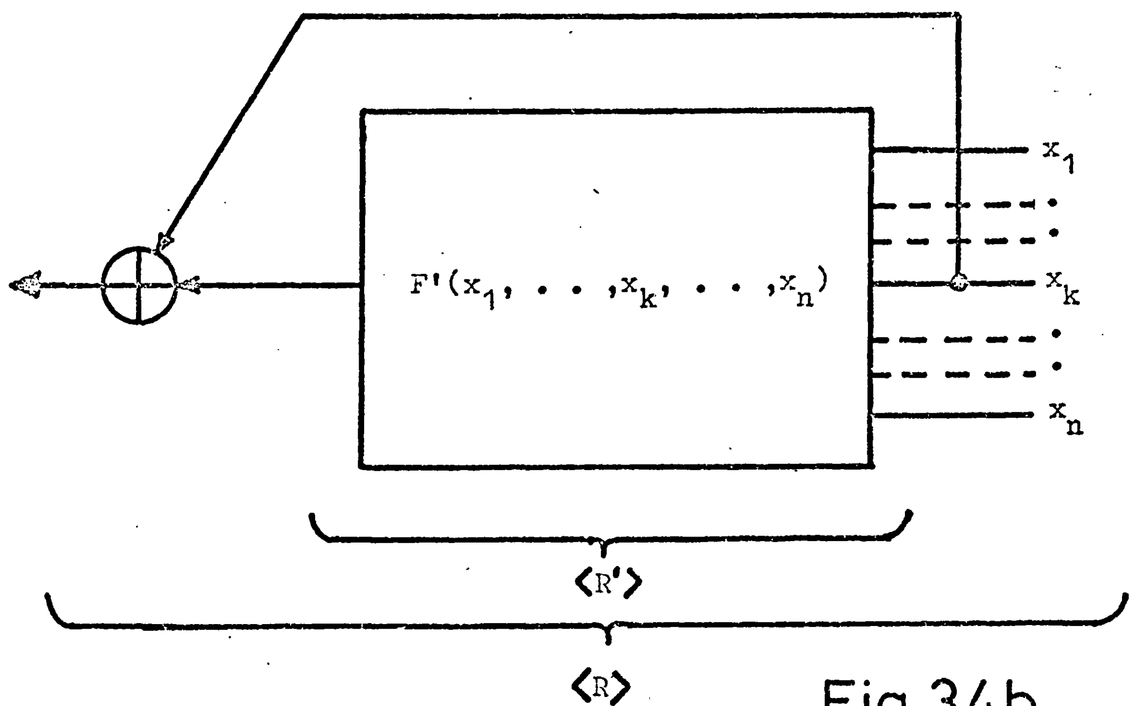


Fig.34b

function of the system remains unchanged.

This operation has resulted in the creation of a feed-forward signal path. If the operation is repeated for two different defining variables then two forward signal paths will be created , and so on.

2.6.4 Significance of Disjoint Spectral Translation.

2.6.4a In Logic Synthesis.

The operation of disjoint spectral translation permits certain functions, which are not translationally equivalent, to be converted one into another. The practical importance of this lies in that it extends the versatility of threshold logic and permits more elegant syntheses in terms of vertex logic.

The implementation of this operation is very straightforward as was shown in the previous section.

2.6.4b In Boolean Function Classification.

Disjoint spectral translation gives rise to a classification of Boolean functions which is more compact than that of translational equivalence (Section 2.5.3a) as is shown below.

The following classification of Boolean functions is proposed :

> A Boolean function $F_1(x_1, \ldots, x_n)$ is classified disjointly-translationally -equivalent to another Boolean function $F_2(x_1, \ldots, x_n)$, of the same order, if $F_1(x_1, \ldots, x_n)$ can be mapped onto $F_2(x_1, \ldots, x_n)$ by the permutation and/or complementation of the defining variables and/or the , perhaps repeated, application of the theorems of spectral translation and/or disjoint spectral translation

---o0o---

Again the tables of canonic function spectra can be drawn up together with optimum syntheses , as in section 2.5.3b . In this case however R_0 is designated the highest magnitude then the first-order coefficients , and so on.

This procedure has been carried out for all Boolean functions, $n \leqslant 4$, and the associated table appears in Appendix 5. The complements of these functions do not appear and are given by Operation 4.

This table shows that the 65,536 functions are classifiable into 8 categories. In practical terms this means that eight logic modules together with the necessary exclusive-OR gates and inverter gates are able to synthesise _any_ Boolean function , $n \leqslant 4$. In fact only seven logic modules are required in practice since function No. 1 in the table corresponds either to a simple connection or a no-connection.

Perhaps more suprising is the fact that only one of the classes of functions is not a threshold function.(Threshold functions are marked 'T').This shows that single threshold gates may be used to synthesise the majority of Boolean functions , $n \leqslant 4$, using the above techniques. Some comment will be made on the synthesis of the non-threshold function , function No. 8 , later.

The fact that this classification is more compact than that of translational equivalence is shown by noting that the latter gives eighteen classes of functions wheras this method gives eight. See also Appendix 3.

2.6.5 _Application to Threshold Logic Synthesis._

Boolean functions which may be converted to threshold functions by the operation of disjoint spectral translation will be said to have threshold functions 'disjointly-embedded' within them.

As an example of a function which contains a disjointly-embedded threshold function consider the function given by the Karnaugh map of Fig. 33 ,p 123 , which has the spectrum

2	2	2	2	6	2	-6	6
R_0	R_1	R_2	R_3	R_4	R_{12}	R_{13}	R_{14}
-6	-2	6	2	-2	-2	6	-2
R_{23}	R_{24}	R_{34}	R_{123}	R_{124}	R_{134}	R_{234}	R_{1234} .

Now it is clear from the tables of positive characteristic vectors , Appendix 4 , that the only threshold function that can be embedded in the above function is that which has a characteristic vector 6 6 6 6 6 . However the above function cannot be converted to this form by spectral translation ,(Operation 5),since R_0 would retain its value '2'. If <u>disjoint</u> spectral translation,(Operation 6), is employed however this problem is overcome as shown below.

1/ Translating $R'_0 = R_4$ under <u>disjoint</u> spectral translation gives

6	6	-2	6	2	-2	-2	2
R'_0	R'_1	R'_2	R'_3	R'_4	R'_{12}	R'_{13}	R'_{14}
6	2	2	-2	2	-6	-6	2
R'_{23}	R'_{24}	R'_{34}	R'_{123}	R'_{124}	R'_{134}	R'_{234}	R'_{1234}

2/ Translating $R''_2 = R'_{23}$ under spectral translation (Operation 5) gives

6	6	6	6	2	-2	-2	2
R''_0	R''_1	R''_2	R''_3	R''_4	R''_{12}	R''_{13}	R''_{14}
-2	-6	2	-2	2	-6	2	2
R''_{23}	R''_{24}	R''_{34}	R''_{123}	R''_{124}	R''_{134}	R''_{234}	R''_{1234}

3/ Translating $R'''_4 = R''_{24}$ under spectral translation (Operation 5) gives

6	6	6	6	-6	-2	-2	2
R'''_0	R'''_1	R'''_2	R'''_3	R'''_4	R'''_{12}	R'''_{13}	R'''_{14}
-2	2	2	-2	2	2	2	-6
R'''_{23}	R'''_{24}	R'''_{34}	R'''_{123}	R'''_{124}	R'''_{134}	R'''_{234}	R'''_{1234}

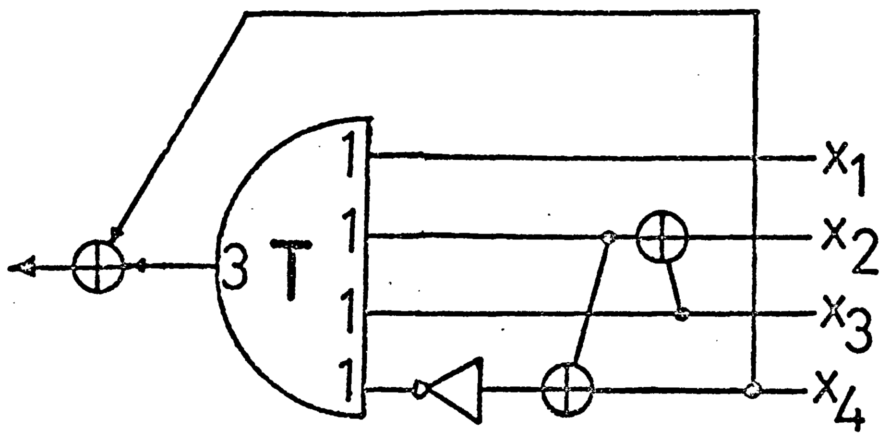


Fig. 35

The first (n+1) spectral coefficients of this function have a magnitude of 6 which characterises it as a threshold function. Computing the gate parameters in the usual way , see Section 2.5.4a , and implementing the above translations in terms of exclusive-OR gates gives the circuit of Fig. 35.

It has been shown , see the previous section , that the majority of fourth-order Boolean functions may be synthesised by using both spectral translation and disjoint spectral translation.

A possible method for the synthesis of functions which do not have threshold functions embedded or disjointly embedded within them is to divide the function into two parts , $x_k \cap F(x_1, \ldots, x_k, \ldots, x_n)$ and $\bar{x}_k \cap F(x_1, \ldots, x_k, \ldots, x_n)$, and to apply the above synthesis procedures to each of these functions in turn. Since these functions do not intersect in n-space the resultant syntheses may be OR-ed together. In the case where this procedure produces another function which does not have an embedded threshold function the division is repeated in terms of another defining variable.

Consider the function of Fig. 36a which does not contain a threshold function.(It falls into canonic class 8 Appendix 5). Suppose that this function is divided as $x_1 \cap F(x_1,x_2,x_3,x_4)$ and $\bar{x}_1 \cap F(x_1,x_2,x_3,x_4)$. See Fig. 36b and Fig. 36c respectively.If the syntheses of these two functions are carried out in the usual way the circuit of Fig. 36d results.

It can be shown that <u>any</u> Boolean function can be synthesised in this way. This follows from the fact that if this division procedure is repeated exhaustively each true minterm will ultimately be extracted separately. Now a function having only one true minterm is always linearly-separable. (A threshold function).

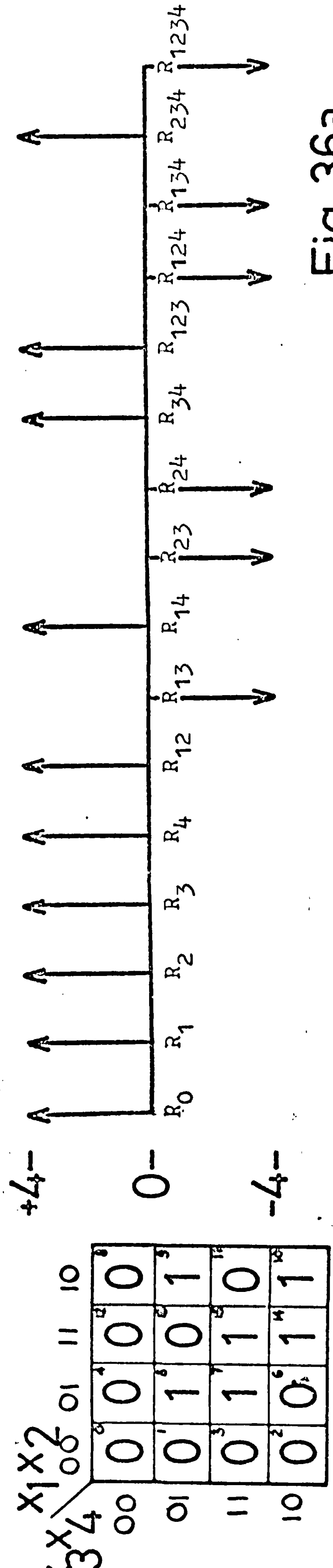


Fig. 36a

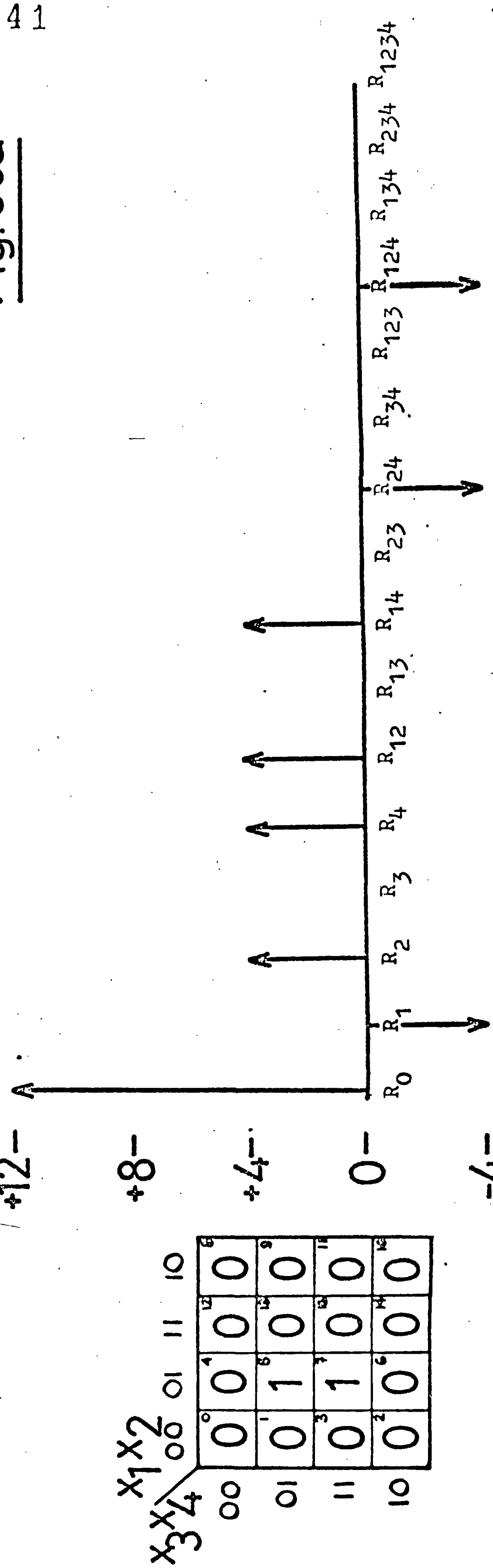


Fig. 36b

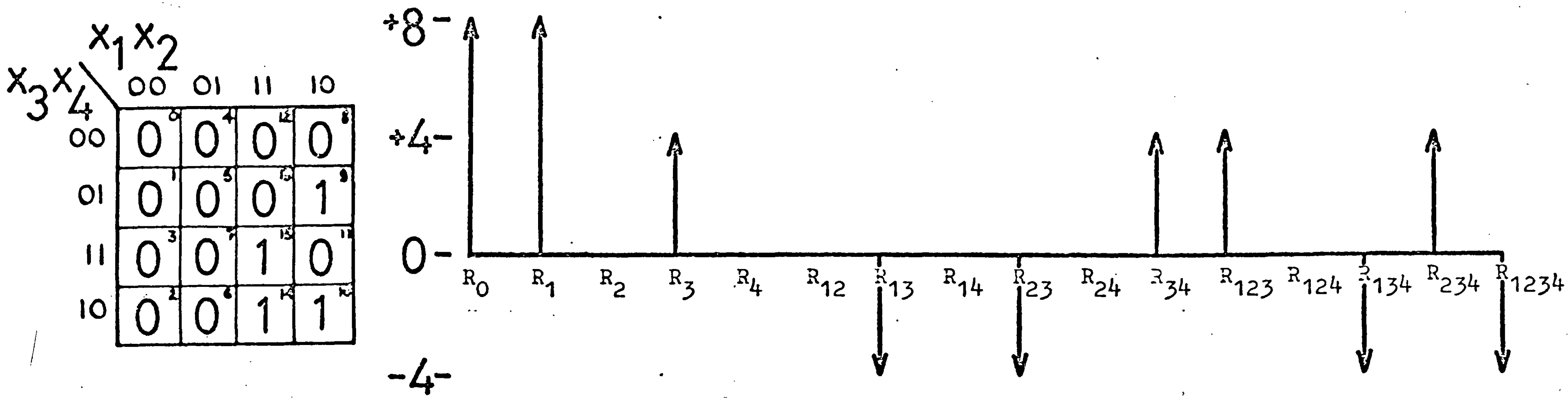


Fig. 36c

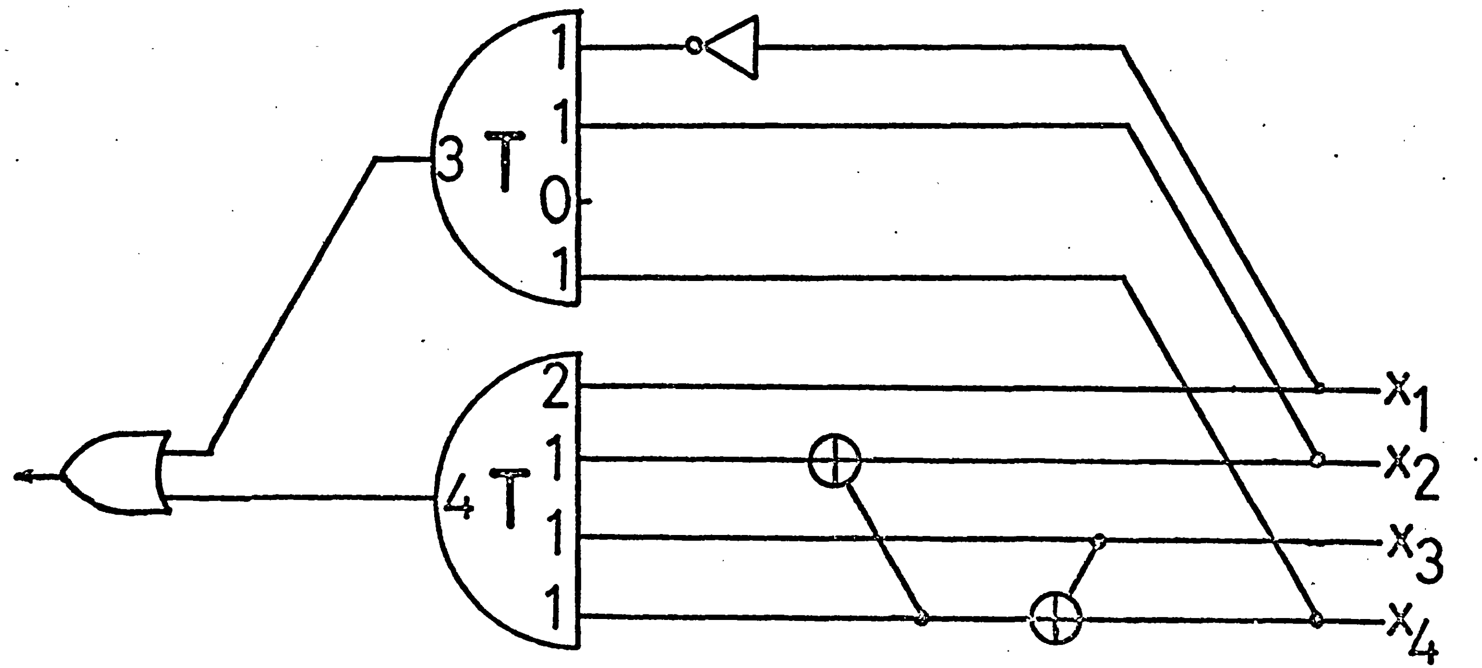


Fig.36d

2.6.6 Application to Vertex Logic Synthesis.

It has been found in practice that the application of disjoint spectral translation often gives a more elegant synthesis than the operation of spectral translation. This however is not always the case. At present the criteria which determine if the use of disjoint spectral translation will give an optimum solution are not known.

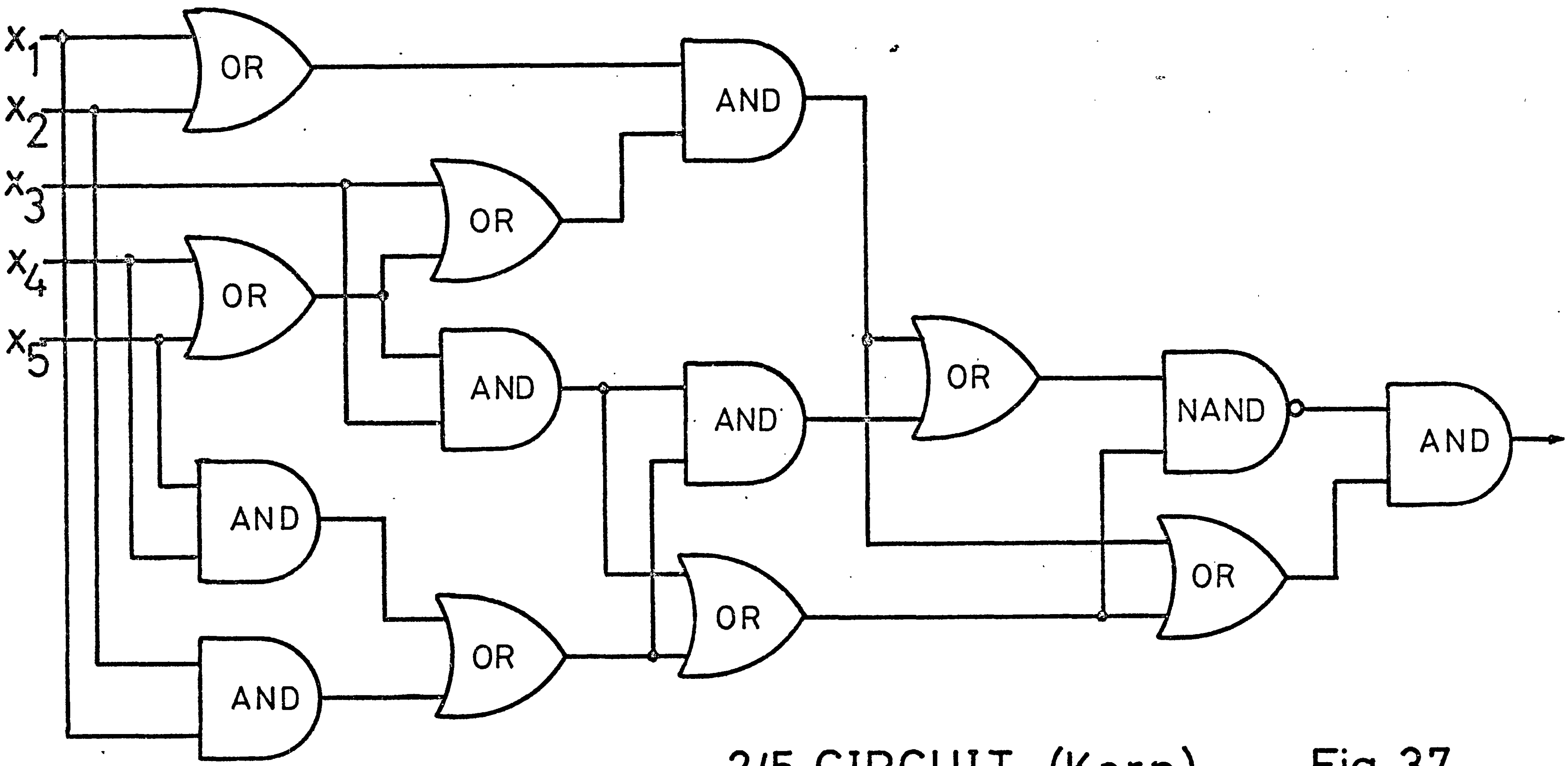
As an interesting example of a case where disjoint spectral translation may be used to advantage consider a 2 out -of 5 circuit. A synthesis , which is believed to employ the minimum number of vertex gates has been published by Karp et al , see reference 30 . This is shown in Fig. 37. An attempt to synthesise this function using spectral translation did not show any advantage over the synthesis of Karp, although admittedly only a simple two-level synthesis of the final logic module was attempted . Under disjoint spectral translation however the circuit if Fig. 38 was produced.* This circuit shows a saving of three gates and two interconnections over the circuit of Fig. 37. It should be noted that the circuit produced by the author may still not be minimal since again only a simple two level synthesis of the function produced by translation methods has been attempted. The maximum propagation delay for both circuits is identical.

2.7 A Statistical Synthesis method.

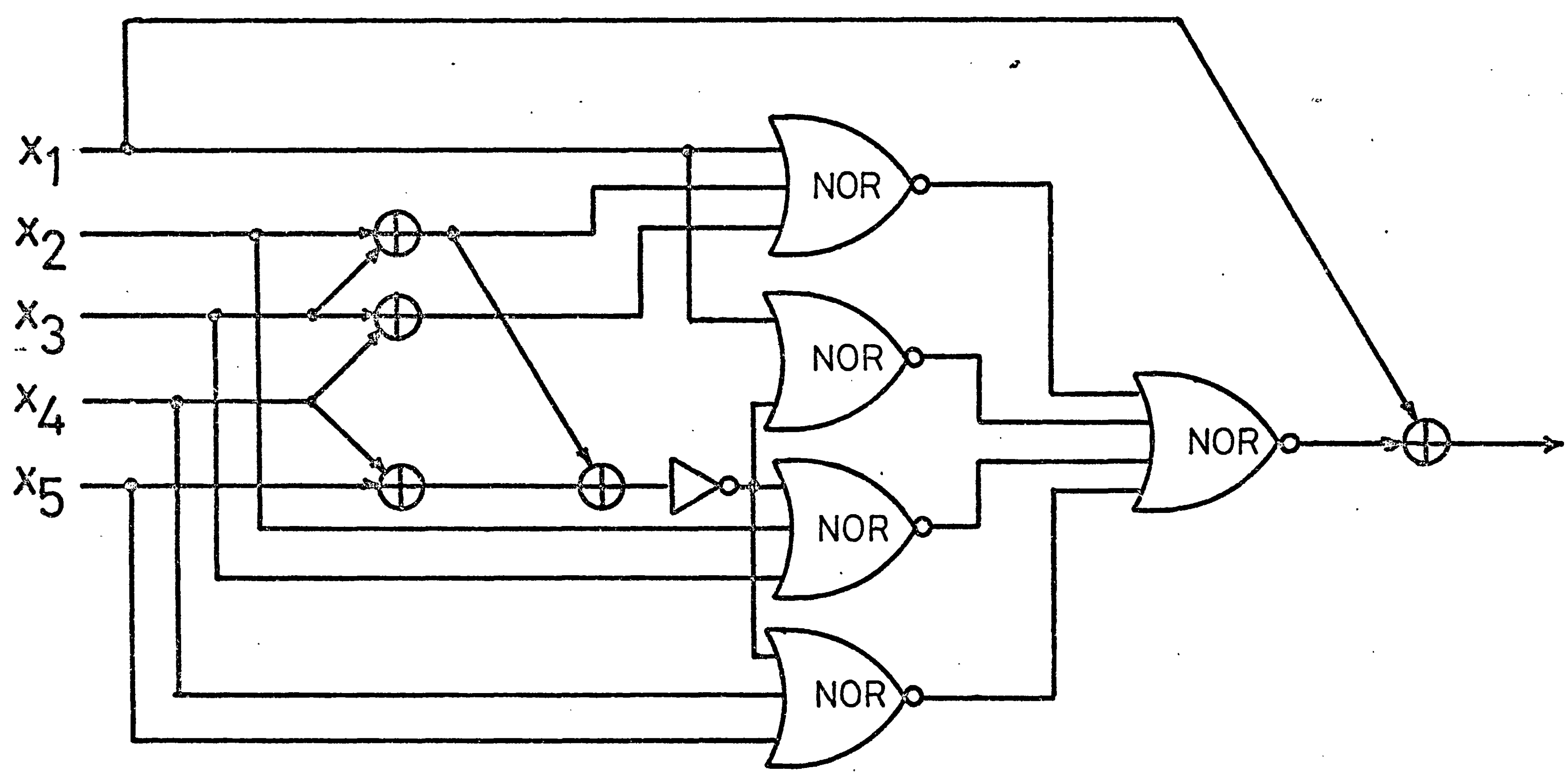
2.7.1 Introduction.

It has been shown by Searle, see reference 25 , and others that the distribution of information in the spectrum of a function is not linear. Indeed in many cases only a small number of the spectral coefficients of a function are necessary to completely define the function, the remaining coefficients being redundant.

* With the aid of the statistical method described in Section 2.7



2/5 CIRCUIT (Karp) Fig. 37



2/5 CIRCUIT (Synthesised Under Disjoint Spectral Translation and Methods of Section 2·7)

Fig. 38

An example of a function having this property is a threshold function where only (n+1) of its coefficients are required.

Now each spectral coefficient is a measure of the correlation between the defining function and each of the Rademacher/Walsh functions. It follows that coefficients of relatively large magnitude indicate that the associated function closely resembles the Rademacher /Walsh functions on which these coefficients are defined.

It seems intuitively correct to suppose that if some of the largest spectral coefficients of a function are known it should be possible to predict the distribution of the minterms of that function on a statistical basis. If this is possible it follows that functions may be synthesised on a statistical basis from only the most significant spectral coefficients, with a consequent saving of both data storage and computer programme execution time.

2.7.2 Spectral Coefficients and the Distribution of Minterms

The transform operation, see Section 2.2.2, may be defined as

$$R_{ij..m} = n_a - n_d \qquad \ldots (2.20)$$

where n_a is the number of agreements between the defining function and the function $x_i \oplus x_j \oplus \ . \ . \ \oplus x_m$, and n_d is the number of disagreements between the defining function and $x_i \oplus x_j \oplus \ . \ . \ \oplus x_m$.

Now
$$n_a + n_d = 2^n \qquad \ldots (2.21)$$

since the defining function must either agree or disagree with $x_i \oplus x_j \oplus \ . \ . \ \oplus x_m$ at all n-tuples.

Substituting for n_d in equation (2.20) gives

$$\begin{aligned} R_{ij..m} &= n_a - (2^n - n_a) \\ &= 2n_a - 2^n \qquad \ldots (2.22) \end{aligned}$$

whence
$$n_a = \frac{R_{ij..m} + 2^n}{2} \qquad \ldots (2.23)$$

Similarly $$n_d = \frac{2^n - R_{ij..m}}{2} \qquad \ldots (2.24)$$

For the special case R_0

$$n_d = \frac{2^n - R_0}{2} = M \qquad \ldots (2.25)$$

where M is the number of true minterms of the function.

For all spectral coefficients with the exception of R_0

$$n_a = T + F \qquad \ldots (2.26)$$

where T is the number of true minterms of the defining function in the space $x_i \oplus x_j \oplus \ldots \oplus x_m = 1$ and F is the number of false minterms of the defining function in the space $x_i \oplus x_j \oplus \ldots \oplus x_m = 0$

Since the space covered by $x_i \oplus x_j \oplus \ldots \oplus x_m = 0$ is $2^n/2$ n-tuples it follows that the number of true minterms in this space is $\frac{2^n}{2} - F$, and thus the total number of true minterms of the defining function ,M, is given by

$$M = T + \left(\frac{2^n}{2} - F \right) \qquad \ldots (2.27)$$

Substituting for F in equation (2.27) from equation (2.26) gives

$$M = T + \frac{2^n}{2} + T - n_a$$

$$= 2T + \frac{2^n}{2} - n_a \qquad \ldots (2.28)$$

Substituting for n_a in equation (2.28) from equation (2.23) gives

$$M = 2T + \frac{2^n}{2} - \frac{R_{ij..m}}{2} - \frac{2^n}{2}$$

$$= 2T - \frac{R_{ij..m}}{2} \qquad \ldots (2.29)$$

Equating (2.29) and (2.25) gives

$$\frac{2^n}{2} - \frac{R_0}{2} = 2T - \frac{R_{ij..m}}{2}$$

then $$T = \tfrac{1}{4} \left(2^n + R_{ij..m} - R_0 \right) \qquad \ldots (2.30)$$

Now T is the number of true minterms in the space where $x_i \oplus x_j \oplus \ldots \oplus x_m = 1$. The number of true minterms of the function is given by equation (2.25).

The importance of this result lies in the fact that the distribution of true and false minterms of a function with respect to any Rademacher/Walsh function can be determined exactly given the corresponding spectral coefficient and R_0.

For example suppose that a fourth-order Boolean function has the spectrum

$$\begin{array}{cccccccc} 10 & 6 & 6 & 2 & 2 & -6 & -2 & -2 \\ R_0 & R_1 & R_2 & R_3 & R_4 & R_{12} & R_{13} & R_{14} \\ \\ -2 & -2 & 2 & 2 & 2 & -2 & -2 & 2 \\ R_{23} & R_{24} & R_{34} & R_{123} & R_{124} & R_{134} & R_{234} & R_{1234} \end{array} .$$

The number of true minterms, from equation (2.25), is given by

$$M = \frac{2^n - R_0}{2} = \frac{16 - 10}{2} = 3 .$$

The number of true minterms in the space where $x_1 = 1$, from equation (2.30), is given by

$$T = \tfrac{1}{4}(2^n + R_1 - R_0)$$
$$= \tfrac{1}{4}(16 + 6 - 10)$$
$$= 3$$

Similarly the number of true minterms in the space where $x_3 \oplus x_4 = 1$ is given by

$$T = \tfrac{1}{4}(2^n + R_{34} - R_0)$$
$$= \tfrac{1}{4}(16 + 2 - 10)$$
$$= 2 \qquad \text{and so on.}$$

Appendix 1 shows all the fourth order Rademacher/Walsh functions plotted on Karnaugh maps.

Now it is of interest to be able to calculate the number of true minterms occuring in spaces corresponding to the intersections of different Rademacher/Walsh functions in order that the complete distribution if true and false minterms may be established. For example if , for a fourth order Boolean function, it is known that the space $x_1 \cap (x_2 \oplus x_3) = 1$ contains four true minterms then, since this space contains only four n-tuples, it follows that $x_1 \cap (x_2 \oplus x_3)$ is a factor of the defining function. See Fig. 39.

It is possible to statistically predict the distribution of true and false minterms at the n-tuples corresponding to the intersection of two or more Rademacher/Walsh functions by using the statistical theory of expected values.

2.7.3 Expected Values.

Suppose that a random set of objects are classified under two independent catagories and that the number of objects falling into each catagory is noted. The number of objects , on average ,falling into both catagories is then given by

$$\hat{e} = \frac{T_1 \times T_2}{M} \qquad \ldots (2.31)$$

where T_1 is the number of objects falling into the first catagory , T_2 is the number of objects falling into the second catagory and M is the total number of objects.(It is assumed that all objects fall into one or other of the catagories). $\hat{e}$ is called an estimated value , see reference 31.

If the objects are classified under three independent catagories then the number of objects falling into all three catagories is then , on average,

$$\hat{e} = \frac{T_1 \times T_2 \times T_3}{M^2} \qquad \ldots (2.32)$$

and so on.

The same theory may be applied , with restrictions, to the estimation of the number of true minterms of a randomly selected Boolean function which lie in a space defined by two or more linearly independent functions.[+]

[+] Personal communication Dr.C.Chatfield. Univ. Bath 1972

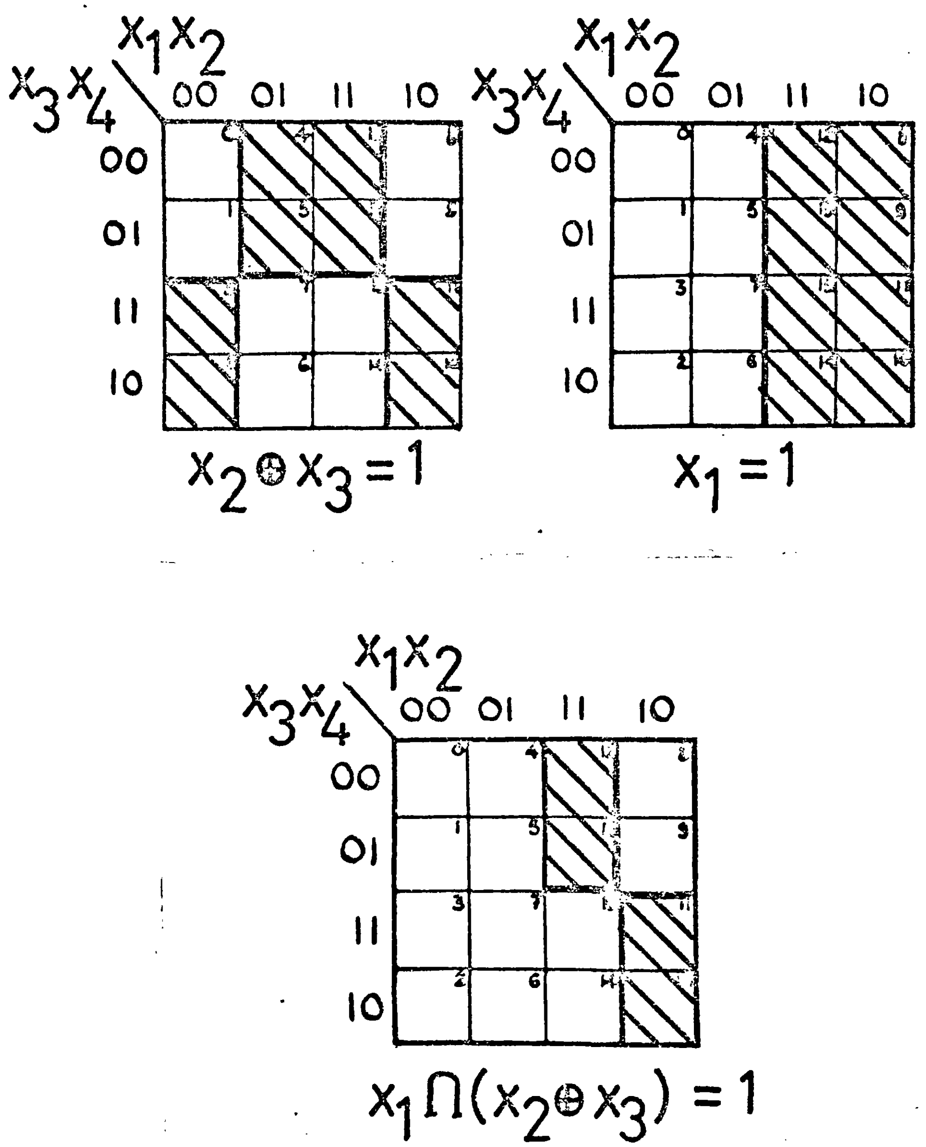
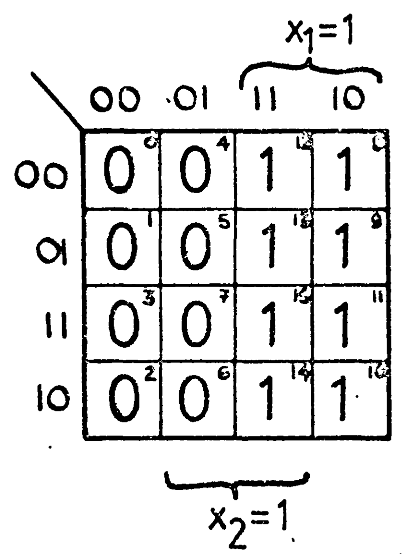


Fig 39



$T_1 = 8$, $M = 8$. T_2 must take value 4

Fig. 40

Suppose that in a Boolean function it is known that the total number of true minterms , M , is 7 ; the number of true minterms , T_1 , in the space $x_1 = 1$ is 7 , and the number of true minterms, T_2 , the space $x_2 \oplus x_3 = 1$ is 4. This data may be derived from the spectrum of the function as previously described. Since the functions x_1 and $x_2 \oplus x_3$ are linearly independent , see Appendix 2 , the estimated number of true minterms in the space $x_1 \cap (x_2 \oplus x_3)$ is given from equation (2.31) by

$$\hat{e} \simeq \frac{7 \times 4}{7}$$

$$\simeq 4 .$$

If this function is fourth order the space corresponding to the intersection of these two functions occupies only 4 n-tuples, thus on average $x_1 \cap (x_2 \oplus x_3)$ can be expected to be a factor of the defining function.

Unfortunately this estimated value is only approximate* because although the functions x_1 and $x_2 \oplus x_3$ are linearly independent the results T_1 and T_2 are not mutually exclusive . This árises from the fact that a finite n-space is being considered. The fact that two such tests, T_1 and T_2 , are in fact related can be shown by the extreme example of Fig. 40 . The total number of true minterms is 8 and the number of true minterms in the space $x_1 = 1$ is also 8. It follows that the number of true minterms in the space $x_2 = 1$ must be 4 . That is , the last result may be predicted from the two previous results ; the measurements are therefore not mutually exclusive. This is in effect a re-statement of the fact that the information about a function is not evenly distributed about the spectral coefficients of that function.

---

* A method of evaluating $\hat{e}$ exactly is known but is very complex and is not suitable for implementation on the digital computer.

In practice the statistic $\hat{e}$ has been found sufficiently accurate for it to be employed in the synthesis method described in the next section. Nevertheless further research is warranted to investigate the general relationships between the exact and approximate forms of $\hat{e}$.

2.7.4 The Procedure.

The method of Boolean function synthesis using the approximate statistic $\hat{e}$ is now given by means of an example.

Consider the fourth-order Boolean function of Fig. 41a. The spectrum of this function is

$$\begin{array}{cccccccc} 6 & 6 & 2 & -2 & 2 & 2 & -2 & 2 \\ R_0 & R_1 & R_2 & R_3 & R_4 & R_{12} & R_{13} & R_{14} \end{array}$$

$$\begin{array}{cccccccc} 2 & -2 & 2 & 2 & -2 & 2 & -10 & 6 \\ R_{23} & R_{24} & R_{34} & R_{123} & R_{124} & R_{134} & R_{234} & R_{1234} \end{array}$$

Step 1

Choose a sub-set of four of the most significant of the spectral coefficients whose defining Rademacher/Walsh functions form a Basis , see Section 2.5.5 and Appendix 2 .

A suitable sub-set is *

$$\begin{array}{cccc} -10 & 6 & 2 & -2 \\ R_{234} & R_1 & R_2 & R_3 \end{array}$$

Step 2

Compute the number of true minterms of the function from equation (2.25)

$$M = \frac{2^n - R_0}{2} = \frac{16 - 6}{2}$$

$$= 5 \quad .$$

---

* Note that the apparently more significant sub-set

$$\begin{array}{cccc} -10 & 6 & 6 & 2 \\ R_{234} & R_{1234} & R_1 & R_2 \end{array}$$ does not define a basis since $|\Lambda| \neq 1$, see Section 2.5.5 .

Step 3

Compute the number of true minterms in the spaces corresponding to the functions on which the basis has been defined using equation (2.30).

Draw up a table showing the result together with the basis functions.

Spectral Coeff.	Value	Basis Function	T
R_{234}	-10	$(\overline{x_2 \oplus x_3 \oplus x_4})$	5
R_1	6	x_1	4
R_2	2	x_2	3
R_3	-2	$\bar{x}_3$	3

Note that in the case where a spectral coefficient is negative the basis function is complemented and T is evaluated for the corresponding value of R made positive. In the case of R_{234} above the result is interpreted as there being 5 true minterms lying in the space defined by $(\overline{x_2 \oplus x_3 \oplus x_4})=1$. Similarly 3 true minterms lie in the space $\bar{x}_3=1$.

Step 4

Find any factors of the function which occupy $\frac{2^n}{2}$, (8), n-tuples .

Since this function contains only 5 true minterms no such factors exist.

Step 5

Find any factors of the function which occupy $\frac{2^n}{4}$, (4), n-tuples.

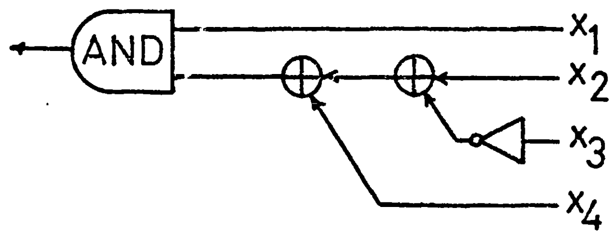
A factor space of 4 n-tuples corresponds to the space defined by the intersection of any two of the functions

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	1
01	0	1	1	0
11	0	0	0	1
10	0	0	1	0

GIVEN FUNCTION

Fig. 41a

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	1
01	0	0	1	0
11	0	0	0	1
10	0	0	1	0

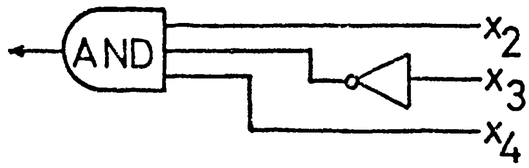


$$x_1 \cap (\overline{x_2 \oplus x_3 \oplus x_4})$$
$$= x_1 \cap (x_2 \oplus \overline{x_3} \oplus x_4)$$

FIRST FACTOR

Fig. 41b

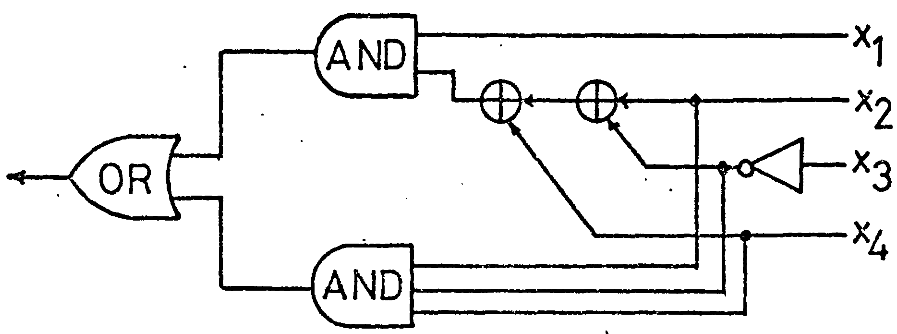
$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0



$$x_2 \cap \overline{x_3} \cap x_4$$

SECOND FACTOR

Fig. 41c



FINAL CIRCUIT

Fig. 41d

of the chosen basis. The pair of functions which correspond to the highest vaues of T are first chosen since these give the (statistically) highest probability of finding 4 true minterms at their intersection.

Choosing the space $(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_1 = 1$ and calculating the estimated average number of true minterms in this space ,from equation (2.31) gives

$$\hat{e} \simeq \frac{T_1 \times T_2}{M} \simeq \frac{5 \times 4}{5}$$

$$\simeq 4$$

It is therefore expected that 4 true minterms exist in this space . In fact this is so , see Fig. 41b.

$(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_1$ is therefore a factor of the given function.

If this procedure is repeated for the next two most significant basis functions , $(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_2$, $\hat{e}$ is found to be $\frac{5 \times 3}{5} = 3$. This is interpreted as a small chance of finding 4 true minterms at the intersection space.

In practice , for fourth-order functions ,having embedded or disjointly-embedded threshold functions , see Sections 2.5.3b and 2.6.4b ,if the ratio $\hat{e}$/(No. of n-tuples in intersection space)$\geqslant$ .9 then the function defining the space is always a factor. This result is empirical and the equivalent result for functions of higher than fourth-order is not known.

In the case of the function under consideration no further factors occupying $\frac{2^n}{4}$ n-tuples can be found.

<u>Step 5</u>

Find any factors which occupy $\frac{2^n}{8}$,(2), n-tuples.

A factor space of 2 n-tuples corresponds to the space defined by the intersection of any three of the basis functions. Again the functions related to the highest values of T are chosen.

It is important to note that in this case there is no point in considering the space $(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_1 \cap x_2 = 1$ as this is included in the factor space $(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_1 = 1$ which has already been found.

The space next most likely to be a factor space is given by $(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_2 \cap \bar{x}_3 = 1$,see previous table.

Computing $\hat{e}$ for this space , from equation (2.32) gives

$$\hat{e} \simeq \frac{T_1 \times T_2 \times T_3}{M^2} \simeq \frac{5 \times 3 \times 3}{5 \times 5}$$

$$\simeq 1.8$$

That is , the average number of true minterms in this 2 n-tuple space is , on average , approximately 1.8 .

The ratio $\left\{\hat{e} \text{ / (No. of n-tuples in intersection space)}\right\} = \frac{1.8}{2}$

$$= .9$$

This space is a factor . See Fig. 41c.

In fact all factors necessary to synthesise the function have been found.

<u>Step 6</u>

Design the circuit.

The expression for the second factor must first be simplified.

The following relationships are noted

$$(\overline{x_a \oplus x_b}) = \bar{x}_a \oplus x_b = x_a \oplus \bar{x}_b \qquad \ldots (2.33)$$

$$x_a \cap (x_a \oplus x_b) = x_a \cap \bar{x}_b \qquad \ldots (2.34)$$

Using these relationships the second factor may be simplified as

$$(\overline{x_2 \oplus x_3 \oplus x_4}) \cap x_2 \cap \bar{x}_3 = (x_2 \oplus x_3 \oplus \bar{x}_4) \cap x_2 \cap \bar{x}_3$$

$$= (\overline{x_3 \oplus \bar{x}_4}) \cap x_2 \cap \bar{x}_3$$

$$= (\bar{x}_3 \oplus \bar{x}_4) \cap x_2 \cap \bar{x}_3$$

$$= x_4 \cap x_2 \cap \bar{x}_3$$

Now the first factor : $x_1 \cap (\overline{x_2 \oplus x_3 \oplus x_4})$ may be written as $x_1 \cap (x_2 \oplus \bar{x}_3 \oplus x_4)$.

The implementation of each of these functions appears in Figs. 41b and 41c and the final synthesis is shown in Fig. 41d .

--oOo--

The method described above appears a little tedious but in fact fast interactive designs can be achieved by employing these techniques on the digital computer. The simplification of the factor equations is also readily computable.

2.7.5 Notes on the Method

More research is necessary into gate minimisation criteria for this method and also the significance of the statistic $\hat{e}$ for functions of order $n \geqslant 5$. The following points are noted.

1/ A more elegant synthesis is often obtained if the true minterms of a given factor are removed from the function and the method repeated for the remaining true minterms. This is because the method evaluates the highest common factors irrespective of the number of gates required.

2/ The choice of basis set has a large influence on the number of gates employed in the final circuit.

3/ The method has been employed successfully for the synthesis of functions of up to ninth-order. Because the nature of the statistic $\hat{e}$ is not well known for orders of greater than four each factor is checked , in these cases , by executing the(inverse) fast Walsh transform for the required spectral coefficients. The factors can then be compared with the defining function in the Boolean domain.

4/ The method is difficult to apply to functions which do not have embedded or disjunctively-embedded threshold functions,† see

† For $n \leqslant 4$ such functions are rare.

Sections 2.5.3b and 2.6.4b . For these functions the statistic $\hat{e}$ is very approximate . This follows from the fact that more than (n+1) spectral coefficients are required to define these functions. Some of the required information to compute $\hat{e}$ therefore lies outside of the basis functions on which $\hat{e}$ is computed. It is felt that another statistic may be found which will enable the synthesis of these functions.

2.8 Further Applications.

2.8.1 Multiple-output Synthesis.

When many functions must be simultaneously realized it is clearly advantageous to make the best use of any common factors the functions may have.

If , therefore , spectral translation is to be employed in the synthesis of such a set of equations , it is possible to set aside a logic module which is capable of executing all of the required translations for the set of equations. Now if some of these translations are identical then this module will be simplified. This amounts to the extraction of the common factors of the functions.

It follows that the judicious choice of coefficients to be translated enables the general method of spectral translation to simplify multiple-output synthesis.

Further research is necessary to find the best methods of determining such common factors.

2.8.2 Synthesis of Functions Containing 'Don't Cares'.

So far only functions which are completely specified have been considered. Functions with don't care conditions give rise to spectral coefficients which may take a range of values , but not independently. At present the optimum method of synthesising such functions is not known.

One approach to this problem is to give the don't care minterms the value $\frac{1}{2}$, that is a value half way between the Boolean values 0 and 1. The spectrum of the function may then be evaluated and analysed statistically as shown in Section 2.7 . The don't care minterms may then be set to 0 or 1 in turn , the final selection of values being determined by those values which produce the highest common factors of the function.

Further research is necessary in this area.

2.9 Conclusions.

A matrix transformation technique has been described which enables the Rademacher/Walsh spectrum of any Boolean function to be evaluated. It has been shown that certain pertinent properties of the Boolean function, from which the spectrum is generated, may be established by inspection of the spectrum alone. In particular it is possible to establish if the Boolean function is most easily synthesised with or without the aid of exclusive-OR gates.

Certain known operations in the 'spectral domain' have been described and it has been shown that these operations enable 'equivalent' Boolean functions to be classified and synthesised. In the search for a more powerful method of Boolean function classification two novel operations have been developed which generate elegant syntheses of Boolean functions both in terms of vertex and threshold logic. Moreover these operations have been shown to give rise to a very powerful method of Boolean function classification. A method of minimising the number of gates necessary to implement these operations has been demonstrated.

It has been shown that many Boolean functions are characterised by only a few of their spectral coefficients . In the future this means that it may be possible to specify such functions , especially those having a large number of defining variables , using only a small percentage of the data space required at present.

One of the most important results arising from this investigation is that threshold functions , and therefore threshold logic , play an important role in the composition of Boolean functions. This is especially important in view of the optimised universal threshold gate developed in Chapter 3.

A statistical approach to logic synthesis , using the spectral coefficients of Boolean functions , has been formulated. Although this method is as yet based upon an approximate-estimated-value technique , practical results have been very encouraging. The great advantage of this method is the ease with which certain 'factors' of a given Boolean function may be extracted. Further research is required in this area.

The execution of the Rademacher/Walsh transform may be carried out, without resorting to matrix multiplication , by means of the fast Walsh transform. This enables the spectrum of functions defined upon large numbers of defining variables to be computed at a much higher speed than would otherwise be possible. In future this should enable functions to be synthesised , using the above techniques, which heretofore have been considered too unwieldy.

Clear indications have been given that the above techniques are applicable to partially specified and multi-output systems. There are also indications that the above methods may be applied to general pattern recognition. Unfortunately time has not allowed a full investigation into these topics.

CHAPTER 3.

Other Research Work

3.1 The Application of a Universal Threshold Logic Gate to Digital Circuit Synthesis.

3.1.1 Introduction.

A universal threshold logic gate* developed by Dr. S.L. Hurst , University of Bath , is described. This gate has the advantage that the problems associated with thresholding tolerances, encountered in conventional analogue threshold gate design , have been overcome.

It is shown that , by employing the theory developed in Section 2 ,a simplified version of this gate is sufficient to enable the synthesis of any Boolean function of fourth-order or less.

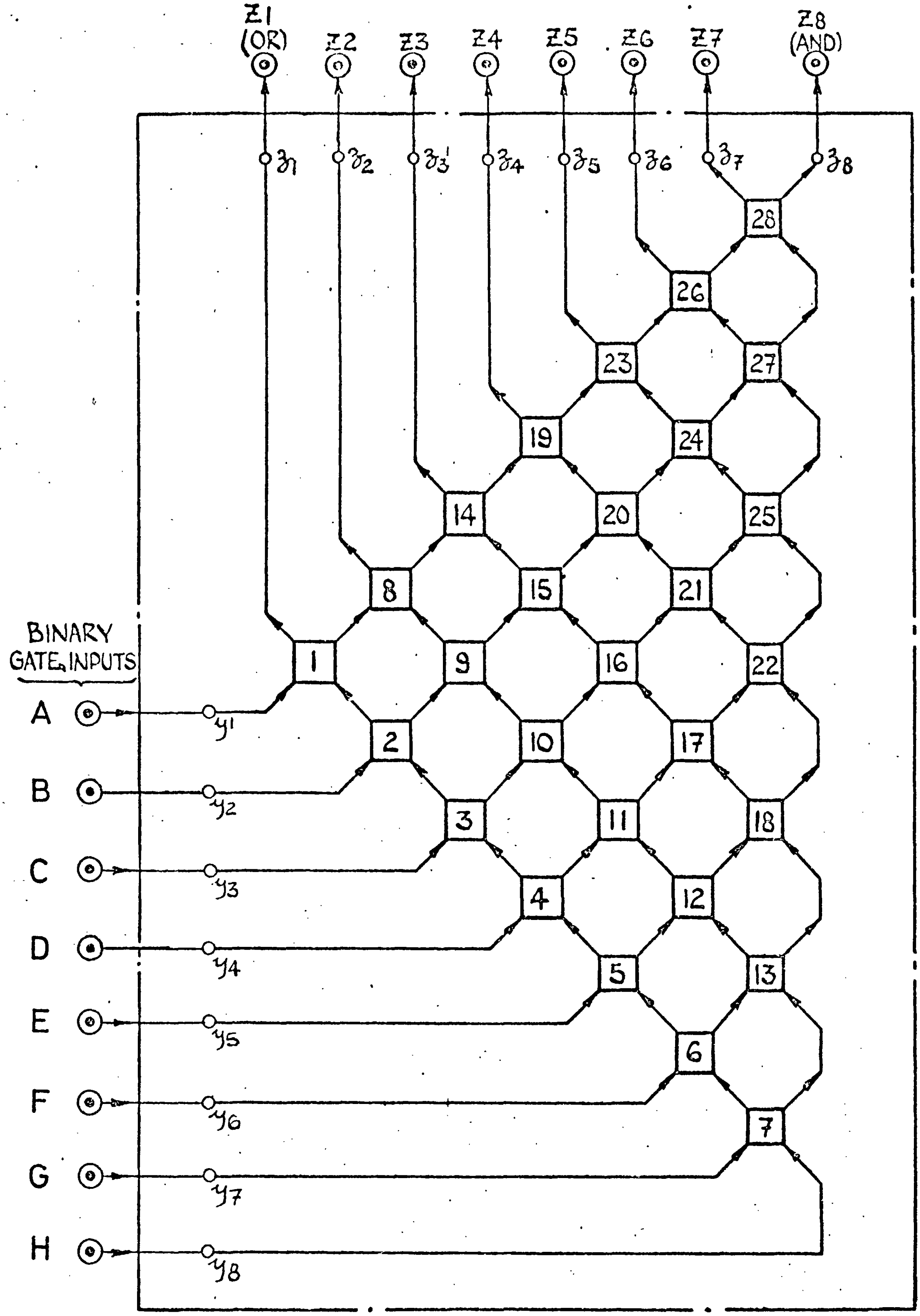
The use of this gate in logic design is expected to provide a considerable cost saving over designs produced by conventional methods.

3.1.2 The Universal Threshold (D.S.T.L) Gate.

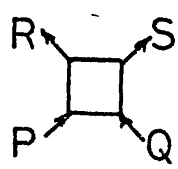
S.L. Hurst , University of Bath , has proposed a Digital-Summation--Threshold-Logic (D.S.T.L) gate of the type shown in Fig. 42 .

In this design each of the eight inputs , labelled A-H, are applied to a logic cell. This row of cells contains conventional digital circuitry and is so connected that if one or more of the inputs A-H have the logical value 1 then a 1 appears at the output z_1. In addition, supposing that N of the inputs A-H have the value 1 , this first row of cells transmits (N-1) values of 1 to the inputs of the next , identical , row of cells. Consequently the second row of cells produces an output of 1 on z_2 if two or more of the inputs

* At the time of writing this design is under consideration for a patent application. The design details should therefore be considered as privileged information.

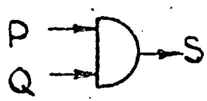


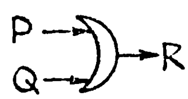
Cell Details, all cells identical :-



$R = [P + PQ],$
$\quad = [P + Q],$

$S = PQ,$

ie 

ie 

ORIGINAL DESIGN
D.S.T.L GATE (Hurst)
Fig. 42

A-H have the value 1 . This process is continued so that the output z_3 is set to 1 if three or more of the inputs A-H have the value 1 , and so on . In practice the number of cells required at each stage reduces by one , see Fig. 42 .

Now this configuration implements a threshold gate where all inputs A-H are weighted 1 and the required output threshold weight may be selected by a suitable connection to one of the outputs z_1-z_8. If an input threshold of weight other than 1 is required , this may be achieved by connecting a suitable number of the inputs A-H together.

In fact , by making suitable input and output connections , any threshold function of order $n \leqslant 4$ may be synthesised using this gate. Because of this property it is termed a Universal threshold gate.

Note that , because digital circuitry is used throughout , no analogue thresholding problems arise.

3.1.3 The Optimised Universal Threshold (D.S.T.L) Logic Gate.

Now , using the theory developed in Section 2 , it is possible to show that a reduced version of the gate of Fig. 42 is sufficient to synthesise any threshold* function of order $n \leqslant 4$.

The positive canonic threshold weighting vectors for $n \leqslant 4$, from Appendix 4 , are

No.	w'_0	w'_1	w'_2	w'_3	w'_4
1	1	0	0	0	0
2	3	1	1	1	1
3	2	1	1	1	0
4	3	2	2	1	1
5	1	1	1	0	0
6	2	2	1	1	1
7	1	1	1	1	1

Consider vector No. 4 .

The corresponding threshold gate input weights are 2,2,1,1 , see equation (2.14) Section 2.5.4a. A total input weighting of 2+2+1+1 = 6 is therefore required for this gate.

*Under disjoint spectral translation and Operation 4.

The output weighting is given by equation (2.15), Section 2.5.4a, as

$$\tfrac{1}{2}\left(\left\{\sum_{j=1}^{n}|w_i|\right\} + w_0' + 1\right) \qquad . . (2.15) \text{ repeated.}$$

$$= \tfrac{1}{2}(6 + 3 + 1)$$

$$= 5.$$

Using Operation 4 Section 2.4 however, it is always possible to render w_0' negative. The minimum output weighting in this case is then

$$\tfrac{1}{2}(6 - 3 + 1)$$

$$= 2.$$

Now if the same analysis is applied to each of the positive canonic weighting vectors of order $n \leqslant 4$ it is found that a universal form of the above gate is sufficient to synthesise them all. That is, a universal logic gate having a total input weighting of 6 and a total output weighting of 2 suffices to synthesise all threshold functions of order $n \leqslant 4$.

This gate is shown schematically in Fig. 43.

The corresponding implementation in terms of D.S.T.L circuitry is given in Fig. 44a. This can be seen to represent a considerable saving in complexity over the circuit of Fig. 42.

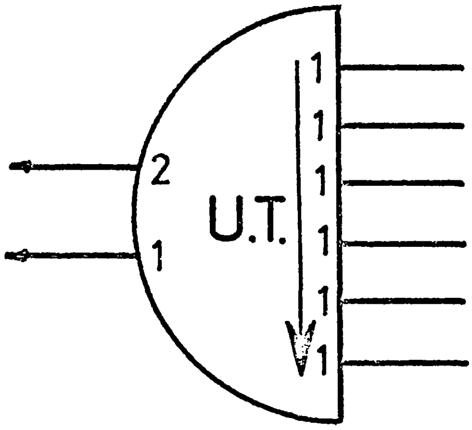
This optimised D.S.T.L. gate has 14 logic gates* and a maximum propagation delay of 6 gates.

3.1.4 Use of the Optimised Gate.

Now it has been shown, see Section 2, that any Boolean function of order $n \leqslant 4$ may be synthesised using threshold logic gates together with the necessary exclusive-OR and inverting gates necessary to carry out the operations described in Section 2.

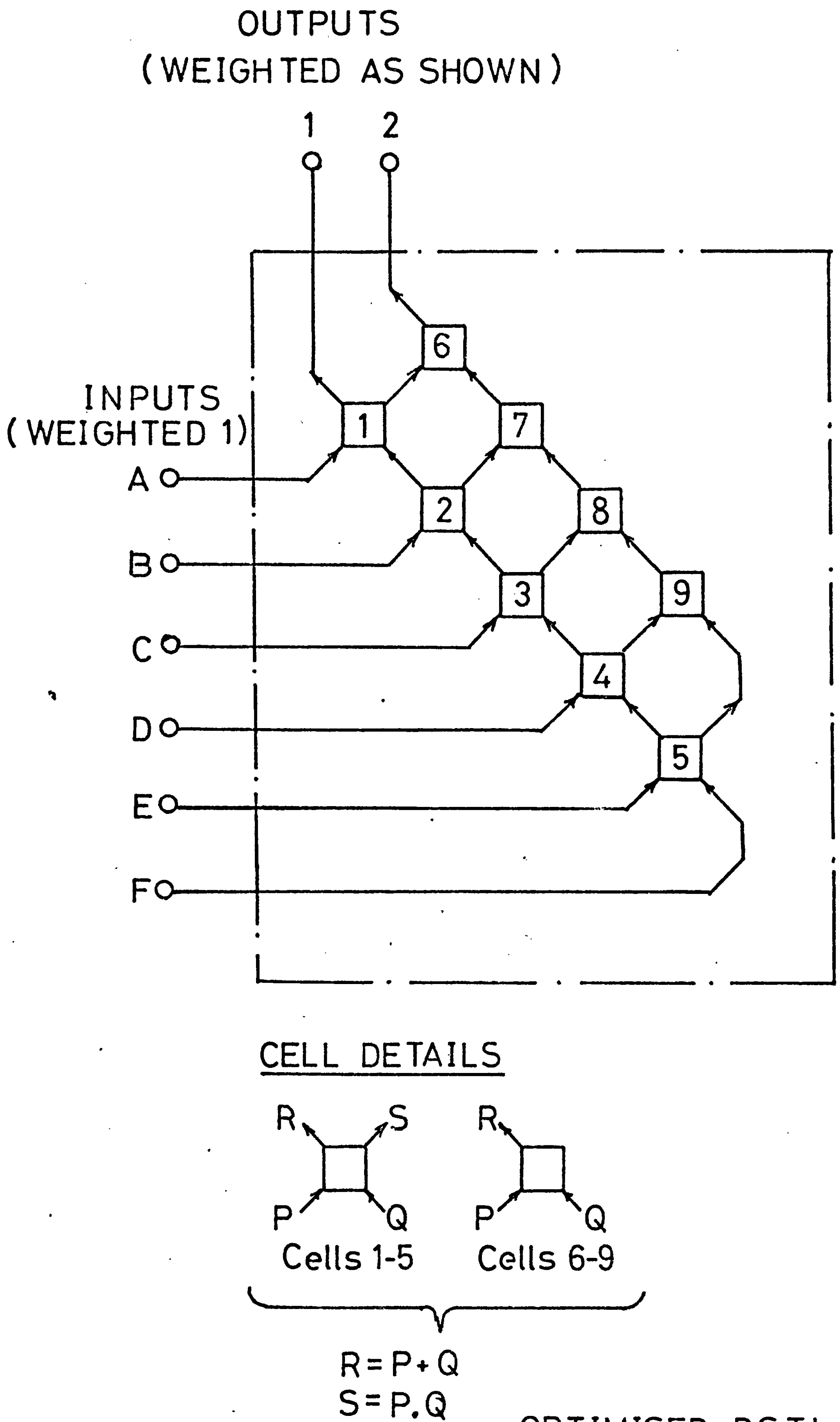
It follows therefore that the optimised universal threshold gate described in the previous section can be used in the synthesis of any Boolean function of order $n \leqslant 4$. Note that functions falling

* 11 logic gates if the 5-input OR gate version is used.



OPTIMISED UNIVERSAL THRESHOLD GATE (Schematic)

Fig. 43



OPTIMISED D.S.T.L GATE

Fig. 44a.

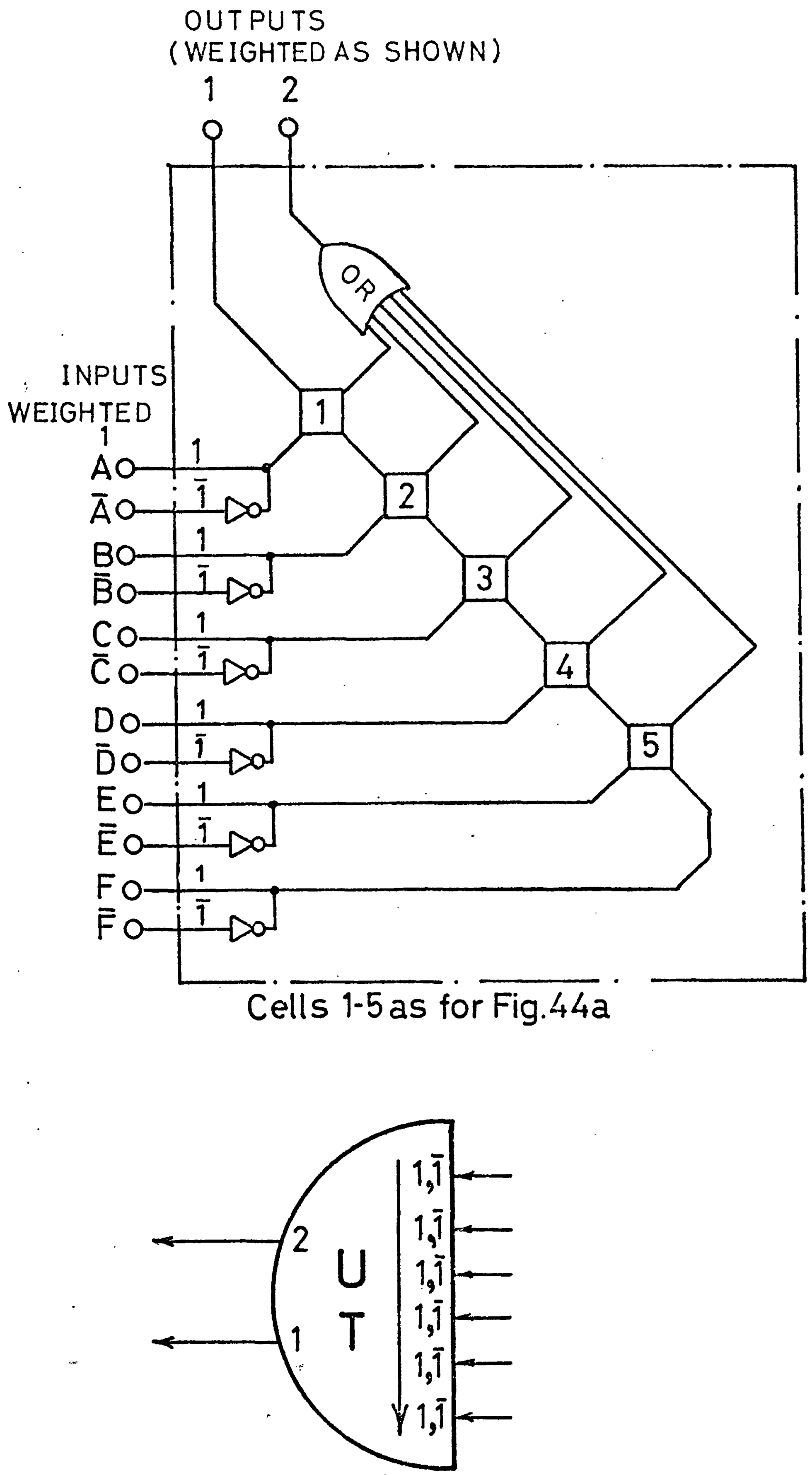
Note: Cells 6-9 may be replaced by one 5-input OR gate.

into the disjoint translationally equivalent class 8 , Appendix 5 , require two such gates , see also Section 2.6.5 . If the synthesis of functions of higher than fourth-order is required this can be accomplished by re-expressing the given function in terms of several fourth-order functions and synthesising each of these in turn. Some further research is necessary to determine the most suitable way of doing this.

If the optimised universal threshold gate in its D.S.T.L form , Fig. 44a, is inspected it will be noted that the propagation delay from input A to the outputs is shorter than from input B to the outputs . Similarly the propagation delay from input B to the outputs is shorter than that of input C to the outputs,and so on. If , say, only four inputs are to be utilised for a particular synthesis it is clear that to minimise the propagation delay only the top four inputs should be employed. The increase of propagation delay with choice of input is shown schematically in Fig. 43 by an arrow.

The method of synthesising functions using this gate follows closely the general methods of synthesis using threshold logic described in Section 2 . The only differences being the use of Operation 4 and the frequent use of disjoint spectral translation to ensure that the input and output thresholds fall within the bounds of the optimised gate.

In practice it is convenient to employ an optimised universal threshold gate with inverted input capabilities. This ensures that no external inverting gates are necessary at the input to the gate to implement negative thresholds,see Section 2.5.4a. Fig.44b shows the optimised gate with this capability . It would also be convenient to have inverted outputs available but this would result in an 18 pin package which is non-standard.



Cells 1-5 as for Fig.44a

OPTIMISED UNIVERSAL THRESHOLD GATE WITH COMPLEMENTED INPUT ($\bar{1}$) CAPABILITY (Schematic)

Fig 44b

Some examples of the use of the gate of Fig. 44b are given in Appendix 6.

In practice it has been found that , in general , the total number of gates and/or interconnections required in a logic synthesis using this gate are considerably smaller than in a synthesis produced by more conventional methods. The cost of implementing such designs is thus smaller than in conventional methods. (This makes the assumption that the D.S.T.L gate can be produced at a reasonable cost. Consultations with integrated circuit manufacturers indicate that this gate can be produced at a cost comparable with that of conventional T.T.L.)

It is envisaged that a cost saving will also result if this gate is used in Large-Scale-Integration circuits.

Because of the advantages outlined above and also because the methods of designing circuits with this gate are straightforward it is hoped that this gate will , in future , become a standard building block for digital circuit fabrication.

3.2 A Cellular Arithmetic Array with Variable Dynamic Range.

3.2.1 Introduction.

The research work described below was carried out† during a general investigation of the properties of iterative arrays and the ways in which such arrays could be represented by Boolean matrices, see Section 1.5.5 .

A particular class of these arrays , often termed cellular arithmetic arrays ,has been investigated by several authors, see references 6,32,33 , and present attractive alternatives to more conventional arithmetic units when extremely fast operation is required. Because these arrays are of an iterative nature they are readily fabricated using Large-Scale-Integration (L.S.I) techniques , and have the additional advantage that they may be readily extended on a modular basis .

A disadvantage of conventional arithmetic arrays is that they produce more significant'bits' in their results than in each of the numbers offered to them. The design described below overcomes this disadvantage and embodies a principle which allows for the multiplication of full floating point numbers.

Following the publication of this design , see reference 34 ,Frecon and Clair showed that arrays of this type may be used in a digital computer design which employs far fewer separate arithmetic instructions than conventional computers . See also reference 35.

A provisional patent for this design was granted in 1970 and a full patent (51122/71), which includes certain additional circuits to extend the versatility of the array , was filed in January 1973.

---

†At the beginning of the research period.

3.2.2 Design Philosophy.

Arithmetic units employing iterative arrays have recently been investigated because of their speed and their ease of fabrication by L.S.I techniques. To take full advantage of the L.S.I methods they consist of two-dimensional arrays of identical logic 'cells', the interconnections between cells being identical and having (ideally) no 'crossovers. All array programming is 'edge-fed' to avoid overlays.

The arrays function asynchronously and achieve a very high computing speed determined solely by the cell and inter-cell propagation delays.

Recent research has centered on integral arithmetic units of this type, see references32,33. The multipliers and dividers developed produce many more significant 'bits' in their results than in the numbers offered to them. In practice this means that truncation and conversion to floating point format must follow, with a corresponding overall speed penalty.

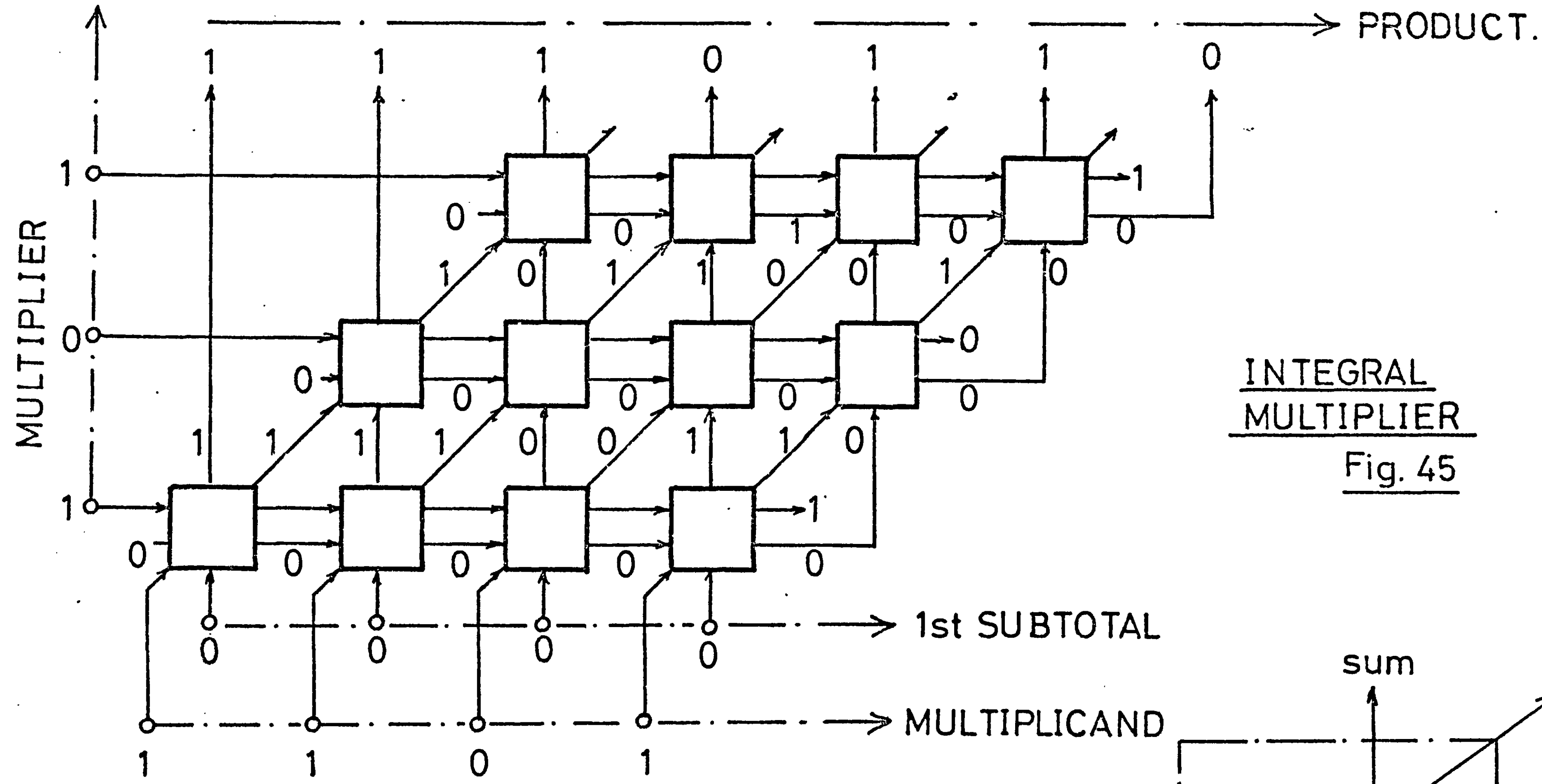
3.2.3 Array Specification.

The multiplier described here overcomes the drawbacks of other systems outlined above and also has other unique features.

Two numbers, each having a binary floating point format, may be multiplied together. The result is expressed as a binary floating point number having the same number of significant 'bits' as the multiplier or multiplicand.

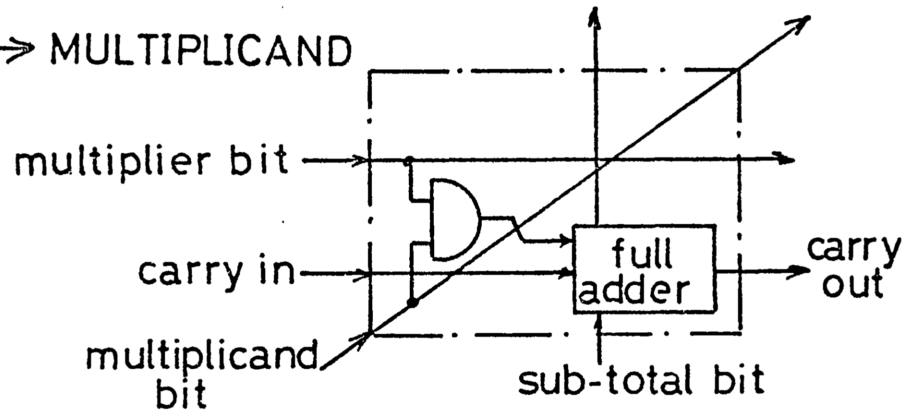
Alternatively, by external programming, the multiplication of two binary integers may be computed to an accuracy determined by the size of the array.

Finally, the number of cells allocated to the calculation of the exponent and the number of cells allocated to the significance part of the result may be varied, within the bounds of the array size.



INTEGRAL MULTIPLIER

Fig. 45



For example , an initial calculation may require an answer of two significant 'bits' and an exponent range of 10 'bits' (2^{1023}) , whereas a second calculation may require 9 significant 'bits' and an exponent range of 3 'bits' (2^7). Both of these calculations may be executed consecutively using the same (12 bit) array of the type described below. The allocation of the cells employed for significance and exponent calculation being determined by external programming. This feature is termed 'variable dynamic range'.

3.2.4 Brief Design Details.

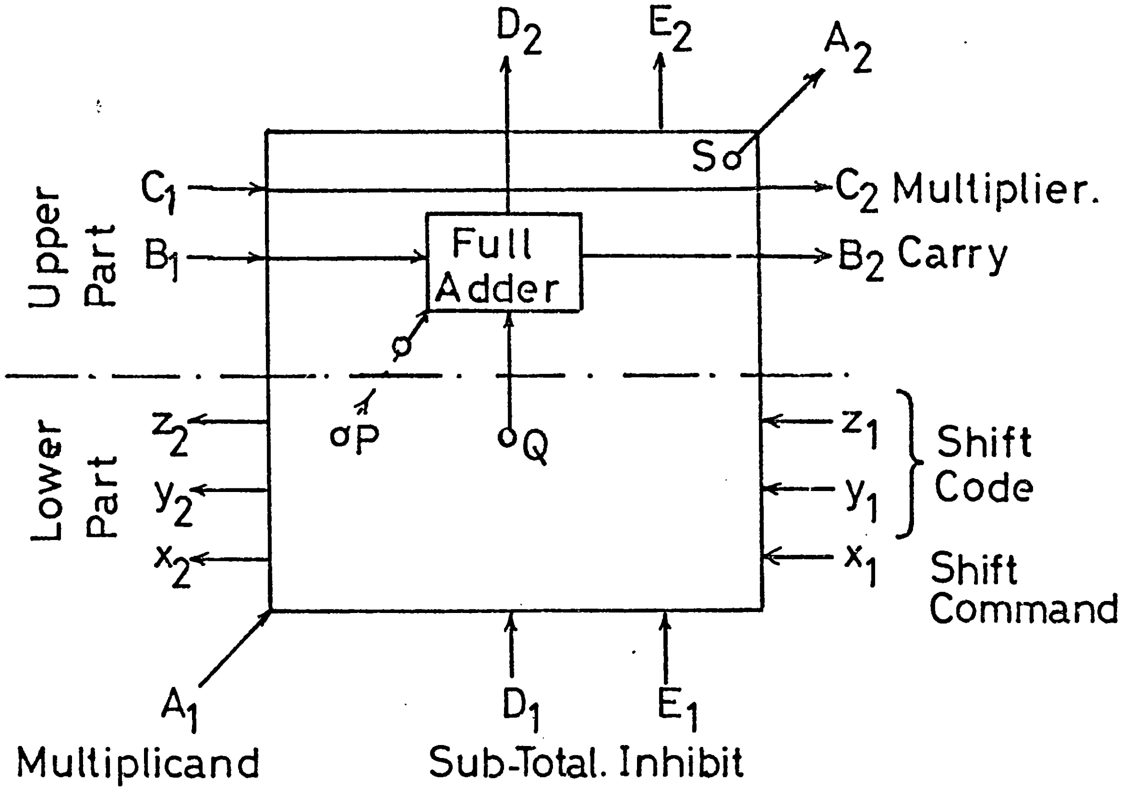
The operation of an integral multiplier is very straightforward and is illustrated by Fig. 45 . The multiplicand is shifted at each stage and then added to a running subtotal if and only if the relevent multiplier 'bit' is 1 . The new subtotal and the shifted multiplicand are then passed on to the next rank of cells. This operation results in the number of significant 'bits' appearing in the subtotal being increased by one at each stage.

Inspection shows that this operation is that of conventional multiplication :

1011	Multiplicand
101	Multiplier
0000	1st. Subtotal
1011	Multiplier bit '1', add multiplicand
1011	2nd. Subtotal
0000	Shift Multiplicand times '0'
01011	3rd. Subtotal
1011	Shift Multiplicand times '1'
110111	4th. Subtotal . (Answer)

Generally the maximum number of 'bits' appearing in the result is the sum of the number of 'bits' appearing in the multiplier and multiplicand.

To reduce the number of significant 'bits' produced, the new design employs cells having a 'return shift' facility, see Fig. 46. Whenever an overflow of the most significant multiplicand 'bit' and/or subtotal carry 'bit' occurs the resultant multiplicand and subtotal



CELL INPUTS/OUTPUTS

Fig. 46

words are shifted one 'bit' by the next rank of cells, the least significant 'bits' being lost. ('Truncation'). This results in a square array. Each time a return shift is carried out the exponent of the result must be increased by one. This is accomplished by means of an identical array of cells ,set aside for this purpose, to the left of the main array . This 'exponent portion ' of the system is set aside by means of external programming.

The logic to accomplish the return shift is contained in the 'lower part' of each cell and was designed using finite-state machine theory, see reference 6 . Specifically , if theinput x_1 , Fig. 46 , is at a 1 then inputs z_1 and y_1 become the new subtotal and multiplicand 'bits' respectively . Outputs z_2 and y_2 carry the original subtotal and multiplicand 'bits' to the next adjacent cell.

The 'upper part' of each cell contains the circuitry of the previously described integral multiplier.

In order that a certain portion of the array may be set aside to calculate the exponent , an inhibit line ,E,is connected to each cell. This line a) inhibits both the shifting of information (by return shift) into the cell and also the shifting of the output multiplicand 'bit',and b) ensures that full addition (in the upper part of the cell) always occurs. A rank of such inhibited cells will act as an adder for a subtotal input D_1, external carry input B_1 and multiplicand input A_1. The first rank of such cells is employed to add the two exponents of the numbers to be multiplied and succeeding ranks add to this result any overflows occuring from the 'significance portion'of the array. This is achieved by a suitable coupling of the output x_2 lines to the external carry inputs. See Fig. 48.

Fig. 47 shows some logic design details of the required cells.

Y_1,X_1 \ A_1,E_1	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	1	0	1	1
10	*	*	*	*

Output 'P'

Y_1,X_1 \ A_1,E_1	00	01	11	10
00	00	00	00	00
01	01	*1	*1	11
11	01	*1	*1	11
10	**	**	**	**

Outputs Y_2,X_2

Z_1,X_1 \ D_1,E_1	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	1	0	1	1
10	*	*	*	*

Output 'Q'

Z_1,X_1 \ D_1,E_1	00	01	11	10
00	00	00	00	00
01	01	*1	*1	11
11	01	*1	*1	11
10	**	**	**	**

Outputs Z_2,X_2

('*' =0/1)

When E=0 and X=1 : $Z_1 \rightarrow Q$ $\quad Y_1 \rightarrow P$

$D_1 \rightarrow Z_2$ $\quad A_1 \rightarrow Y_2$

Otherwise $A_1 \rightarrow P$

$D_1 \rightarrow Q$

P \ E_1,C_1	00	01	11	10
0	00	00	00	00
1	10	11	01	01

S,R

When E=0 : Half/Full addition controlled by multiplier bit C_1

When E=1 : Full addition always occurs, multiplicand bit A_2 always '0'.

Note E=0 represents NO INHIBIT , E=1 represents INHIBIT.

X=0 represents NO SHIFT , X=1 represents SHIFT

CELL LOGIC DETAILS

Fig.47

In use care must be taken to ensure that no overflow from the exponent portion of the array into the significance portion of the array can occur.

Fig. 48 shows an example of the array in use. The inhibit lines have been set to give a significance range of 4 'bits' (2^4-1) and an exponent range of 4'bits' (2^{15}). The numbers appearing within each cell represent the inputs to the upper part of the cell , P,Q , and are the multiplicand and subtotal (left-right) respectively.

3.2.5 Performance.

Since all return shifts depend upon the carry from the previous rank they represent the greatest propagation delay within the array. Since however, the return shifts operate in 'parallel' , that is the return shift from one cell to its neighbour is independent of any other return shifts taking place, the delay per. rank introduced over that of an integral multiplier is that of only two or three gates. In addition a small propagation delay is introduced by the shifting circuitry of the lower part of each cell.

Overall the array can be said to compare favourably with that of a comparable integral multiplier.

3.2.6 The Prototype Array.

A prototype array*has been designed and built which comprises 96 cells arranged,for test purposes, in an array of dimensions 8 by 12 . T.T.L 7400 series D.I.L logic was employed throughout. The logic design for each cell appears in Fig. 49.

The array has been found to function as predicted.

---

* The logic design was carried out by the author. The design was verified using a logic analysis programme ("B.C.A.P"-see 1973 Internal report University of Bath.) The cells were manufactured by Jasmin Electronics Ltd. The assembly and testing were carried out by T.Bond , University of Bath as a final year project. Finances were provided by The Dept. Electrical Engineering, University of Bath.

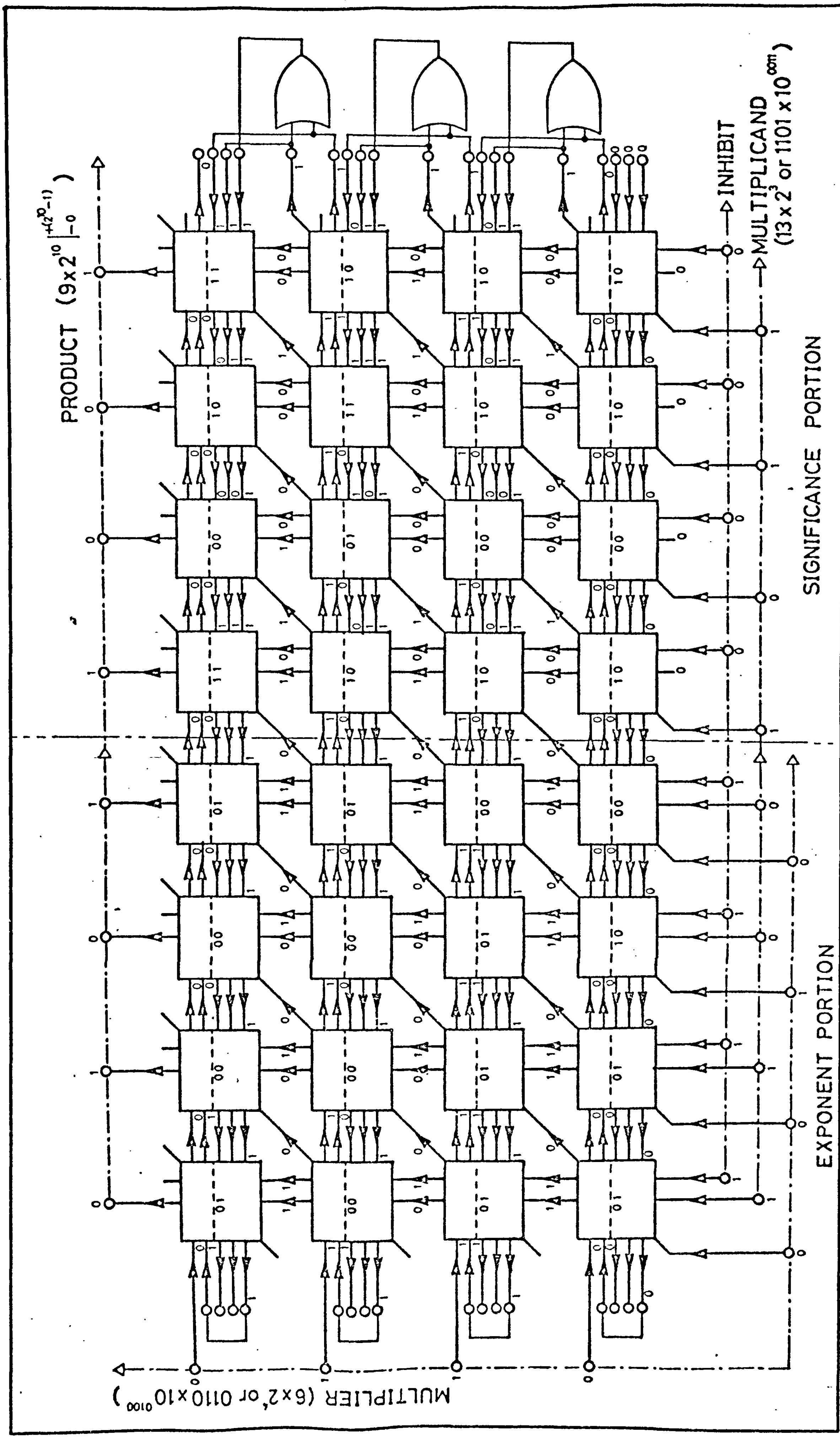
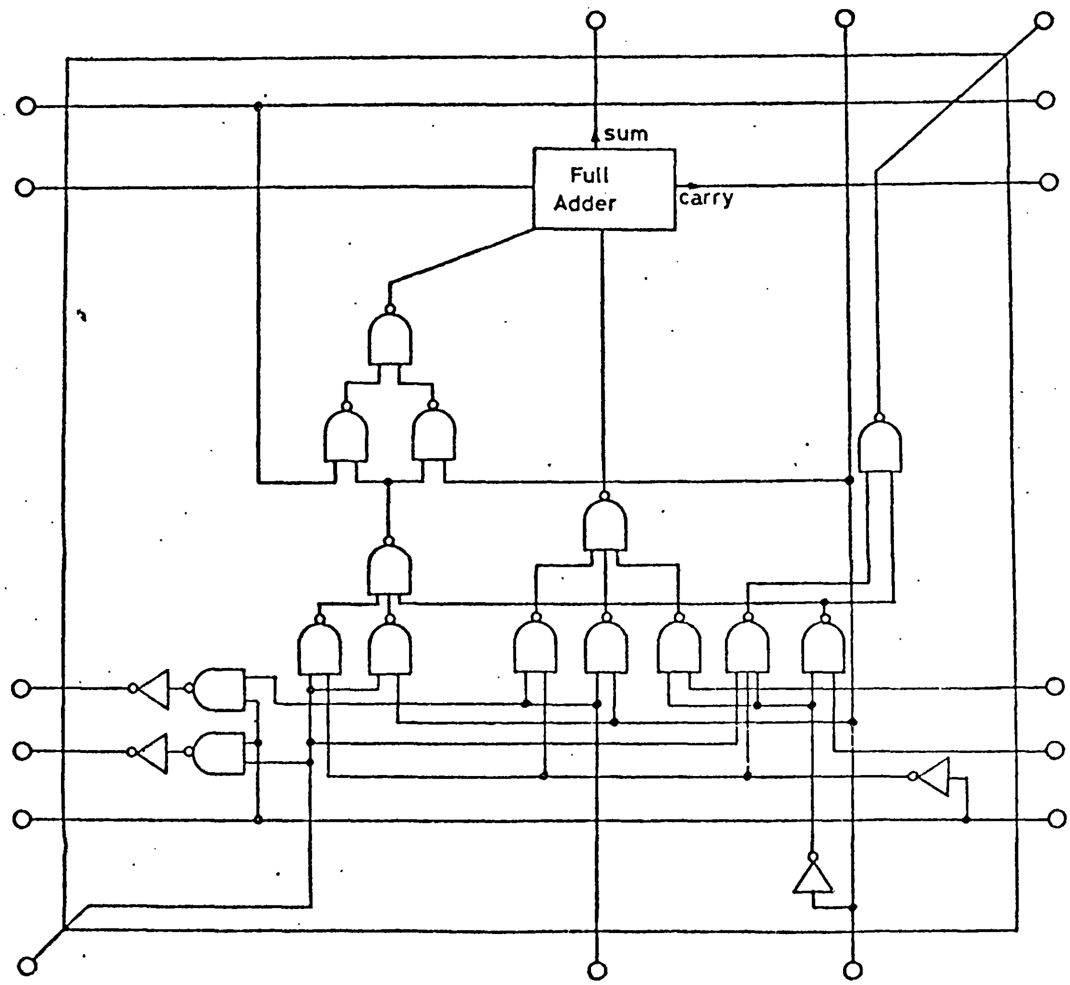

PRODUCT (9x2^10 |+(2^10-1) -0)
SIGNIFICANCE PORTION
EXPONENT PORTION
INHIBIT
MULTIPLICAND (13 x 2^3 or 1101 x 10^0011)
MULTIPLIER (6x2^4 or 0110x10^0100)

Fig.48



CELL DETAILS

All inputs/outputs orientated as shown

NAND

INVERT

Fig. 49

In its floating point configuration the array will function fastest if the binary numbers offered to it are most-significant- 'bit' orientated, that is a 1 appears on the input of the array which corresponds to the most significant 'bit' of the number. This ensures that the shift command , input x_1 , in each row of the array is set up with the least possible delay. Under these circumstances the maximum time for the array to multiply two numbers is given by

$$T = P_0 + (S-1)\, P_1 \qquad \ldots (3.1)$$

where P_0 is the maximum propagation delay through a cell from multiplier 'bit' or subtotal 'bit' input to sum 'bit' or carry 'bit' output.

P_1 is the maximum propagation delay as for P_0 but with shift command instigated.

S is the maximum number of significant 'bits' being processed.

In the prototype array the predicted values for the above were

$P_0 = 60$ nS
$P_1 = 80$ nS and, in the configuration used,
$S = 8$

The expected maximum delay time was therefore $T = 60 + 7\ 80 = 620$ nS.

The measured maximum time was 540 nS. The discrepency is probably accounted for by differences between the manufacturer's estimate of gate propagation delay times (possibly pessimistic) and the delay times of the gates in practice.

The average power consumed by each cell, in the quiescent state , was 0.44 watts. No figures are yet available for power consumption during computation.

If these figures are extrapolated for an array capable of handling numbers of the order : 7 significant digits (decimal), range 10^{99} , then the estimated time for multiplication is 1.9 μS and quiescent power consumption is approximately 350 watts. The time for multiplication represents a considerable saving over modern conventional multipliers.

In practice an array of the size just mentioned would be more economically produced in integrated circuit form , several cells being implemented by one of such circuits. It is unlikely that the whole array would be produced as one integrated circuit because of the difficulties in dissipating the heat produced.

An array of the same size as the one just discussed but employing devices of low power consumption , eg. C.O.S.M.O.S.F.E.T's*, could be produced as one integrated circuit 'chip' and would be an attractive circuit for incorporation in modern'pocket calculators'.

Although the array described in this section does not strictly come under the heading of a 'matrix method' , the investigation of the properties of this , and like, arrays was prompted by the need to fully understand the behaviour of general iterative arrays in the light of Boolean matrix theory. It has therefore been included as a piece of research closely related to matrix methods.

In the final analysis , however, it has been found that the representation of such arrays by Boolean matrices does not facilitate their synthesis for reasons described in Section 1.3.9 ,p 46.

* Complementary-Symmetry Metal Oxide Semiconductor Field-Effect Transistor.

CHAPTER 4

General Conclusions and Recommendations for Further Work.

4.1 General Conclusions.

This thesis has presented some new approaches to logic synthesis by matrix methods.

In Chapter 1 an investigation into the properties of Boolean matrices, of a particular type, was described. It was shown that the properties of the algebra associated with these matrices give rise to a method of analysing a function in terms of its dependence upon any chosen set of its defining variables. The exhaustive application of this technique , using Boolean matrices, was shown to permit the extraction of the prime implicants of several functions simultaneously and to have certain advantages in this respect over the method of Quine-McCluskey. An iterative method for the synthesis of Boolean functions, which generates optimum solutions on an exhaustive search basis , was also developed. This technique enables partially specified systems having multiple outputs to be synthesised using any chosen logic modules as 'building blocks'. Other concepts of general interest were those of pre- and post-multiplicative operators and the possibility of defining 'dependent' functions.

Chapter 2 was concerned with a matrix transformation technique which enables the Rademacher/Walsh transform of any Boolean function to be determined. The choice of this transformation as a tool for logic synthesis arose from a search for techniques of synthesis which do not have an iterative structure and which allow the logic designer both to readily grasp the properties of the system to be designed and also influence the resulting synthesis. It was shown that certain pertinent properties of a Boolean function could be gleaned from a study of the Rademacher/Walsh transform of that function. Certain novel spectral operations were developed which allow elegant syntheses of Boolean functions both in terms of threshold and vertex logic. A straightforward method of gate minimisation was derived. It was also shown that these operations enable Boolean functions to be classified in a very concise way. This classification showed that threshold functions play an important part in the composition of Boolean functions. A novel synthesis method based upon an approximate statistic was proposed. The results of this method are , at present , very encouraging. Further research into this topic is necessary.

Chapter 3 was concerned with the research work arising from the work of Chapters 1 and 2 . Of special interest was the development of an optimised universal threshold gate which , under the operations described in Chapter 2 , is able to synthesise any fourth-order Boolean function having an embedded or disjointly embedded threshold function. Fourth-order functions not falling into this category may be synthesised by using two of such gates. It also follows that functions of order $n > 4$ may be synthesised by several of such gates. It is felt that this gate may , in future , become a standard module for the design of logic circuits since , in practice , it has been found that the use of this gate allows circuits to be designed at a lower cost than is possible at present. The design procedures for the synthesis of Boolean functions using this gate are straightforward , following closely the methods of Chapter 2 .

The Boolean matrix methods of Chapter 1 allow for the representation and synthesis of cascaded logic modules . This property does not seem to be shared by the techniques of Chapter 2 however. It is felt that an investigation into the relationships between these two disciplines may result in an approach to synthesis which embodies the special advantages of both of them.

The fact that , at least in the fourth-order case , many Boolean functions are characterised by only a small number of their spectral coefficients may indicate that,for higher-order functions,it may be possible to completely specify the majority of functions using only a small amount of the data space required at present. For this reason , and also because of the existance of the 'Fast Walsh Transform ' it may be possible to synthesise functions , using the techniques developed in this thesis , of a higher-order than has been attempted using conventional methods.

4.2 Recommendations for Further Work.

1/ The transformation techniques of Chapter 2 , unlike the Boolean matrix methods of Chapter 1 , do not seem to facilitate the representation , and thus synthesis , of cascaded logic modules. A cursory examination of this problem indicates that some form of convolution in the Rademacher/Walsh spectral domain is necessary to represent such cascaded modules. Further investigation is required to establish the relationships between the methods of Chapters 1 and 2 in order that optimal synthesis methods for cascaded logic modules , and indeed finite state machines†, may be established.

2/ The ability of Boolean matrix algebra to define 'dependent' functions warrents further research , see Section 1.3.7 . The property of one function influencing another appears to have applications in adaptive logic systems.

3/ More research is required into the specification of 'don't care' minterms under the Rademacher/Walsh transform. To date this problem has only been given a small amount of consideration.See Section 2.8.2.

4/ It is felt that gate minimisation methods for multi-output logic synthesis under the Rademacher/Walsh transform can be developed with little effort. A theoretical approach to this problem has been given in Section 2.8.1.

5/ The fact that the great majority of fourth-order Boolean functions are characterised by only a small proportion of their spectral coefficients is felt to be very important. It indicates that functions having a large number of defining variables may be specified using a far smaller data space than is required at

† With certain restrictions

present. Specifically , it may be possible to specify most functions by means of the (basis) positions of their most significant spectral coefficients. In addition , under disjoint-translational-equivalence, certain pertinent properties of a Boolean function may be evaluated immediately from the properties of the 'class' in which the function lies. To this end it is important that the disjoint-translational-equivalent classes of functions of order $n \geqslant 5$ should be evaluated. The results given in Chapter 2 for all fourth-order functions (and less) were generated by classifying all the fourth order functions in turn. This process took approximately 17 hours. This method becomes impractical for functions of order $n \geqslant 5$. (The estimated time required for the classification of functions of order $n=5$ on this basis is approximately 100 years !) This problem may be solved by finding the number of functions which may be generated from the (known) canonic characteristic threshold vectors, under disjoint-translational-equivalence, and then instigating a search (on a random basis) for the remaining , non-threshold disjointly-translationally-equivalent ,functions.

6/ For reasons explained in Section 2.7 further research is necessary into the significance of the approximate estimator $\hat{e}$ for functions of order $n \geqslant 5$, and also for functions not having disjointly-embedded threshold functions.

7/ It is known that functions not having disjointly - embedded threshold functions may be synthesised if the function is 'divided' ,see Section 2.6.5. Optimal methods of carrying out this division , and the role that such functions play in the composition of functions of order $n \geqslant 5$, remain to be investigated.

8/ The optimal universal threshold gate was developed towards the end of the research period and only a small amount of time has been devoted to the investigation of its properties. In view of its importance in the low-cost synthesis of logic systems and the ease with which such syntheses may be established , compared to more conventional methods , further research into the automated design of circuits using this gate appears to be of great importance.

4.3 Acknowledgements.

The author is especially indebted to Dr. S.L. Hurst, University of Bath , for his continued assistance and advice throughout the period of research.

The support of the School of Electrical Engineering , University of Bath is gratefully acknowledged.

Many profitable discussions with the following members of staff and post-graduates of the University of Bath also contributed to the research work :

Mr. B. Ireland.	School of Mathematics.
Mr J.D. Martin.	School of Electrical Engineering.
Mr. J. Metcalfe.	Post-graduate. School of Electrical Engineering.
Mr. P.A. Chambers.	Post-graduate. School of Electrical Engineering.

The research was supported by Science Research Council (U.K) Grant B/70/1295.

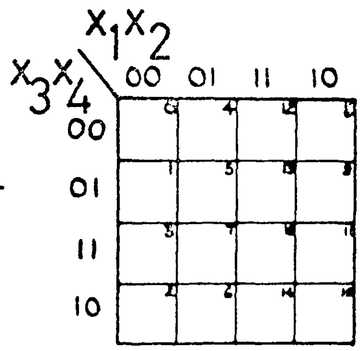
Finally , my most humble thanks are due to my parents who , by their encouragement and support , have made this research possible.

C.R. Edwards.

Univ. Bath. 10 Sept. 73

APPENDIX 1

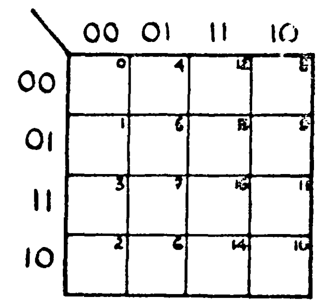
Karnaugh maps of all fourth-order Rademacher/Walsh functions in the range 0,1.



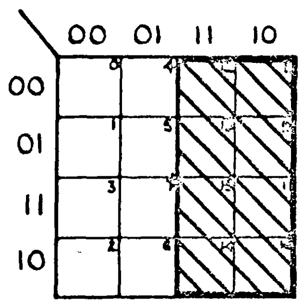
Shaded : TRUE

Blank : FALSE

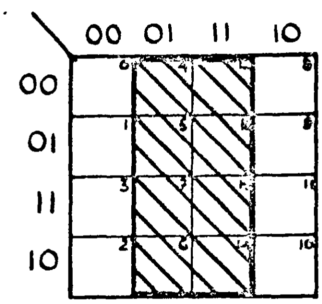
KEY



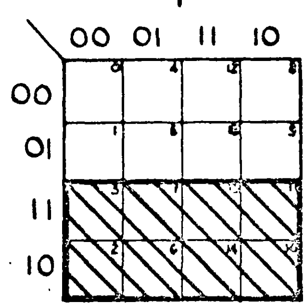
x_0



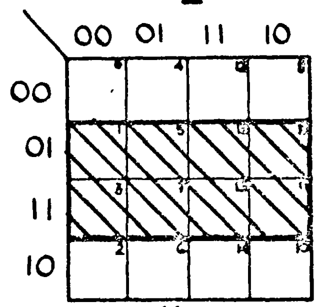
x_1



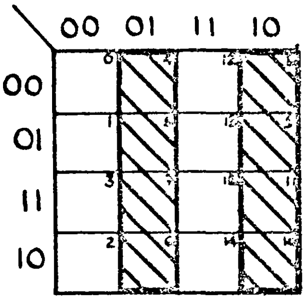
x_2



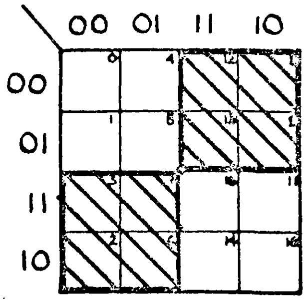
x_3



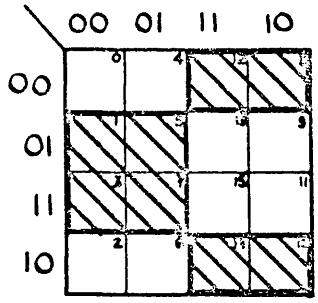
x_4



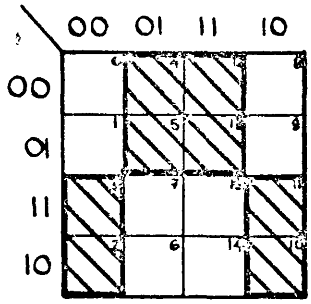
x_{12}



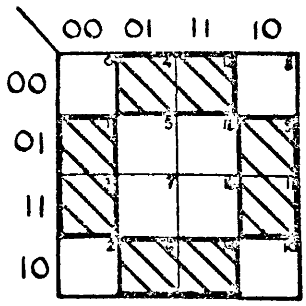
x_{13}



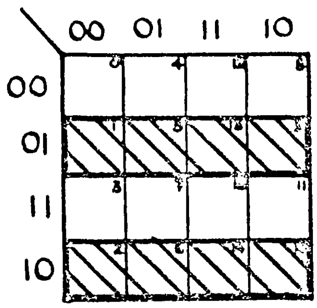
x_{14}



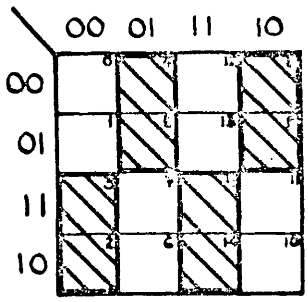
x_{23}



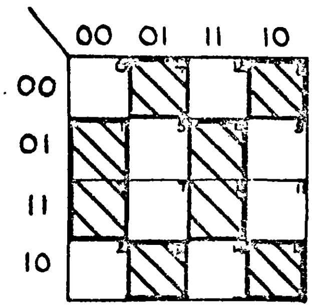
x_{24}



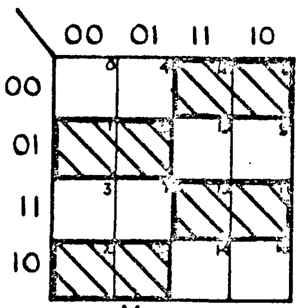
x_{34}



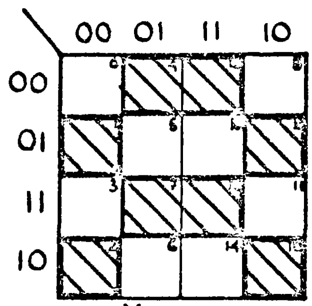
x_{123}



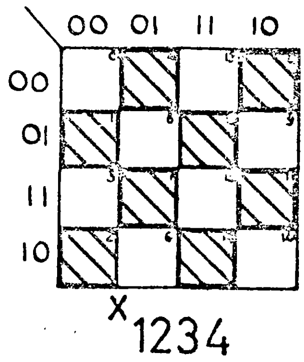
x_{124}



x_{134}



x_{234}



A1.3

APPENDIX 2

The Interpretation of Spectral Translation in terms of Field Theory.

The spectral translation operation concerns itself with the generation of a new function $F'(x_1, \ldots, x_k', \ldots, x_n)$ from a given function $F(x_1, \ldots, x_k, \ldots, x_n)$, where x_k' has the form $x_k \oplus \left\{ x_a \oplus x_b \oplus \ldots \oplus x_h \right\}$ and $F(x_1, \ldots, x_k, \ldots, x_n) \triangleq F'(x_1, \ldots, x_k', \ldots, x_n)$. It is required to establish that a unique function $F'(x_1, \ldots, x_k', \ldots x_n)$ is always generated under these constraints. If this is so the validity of the spectral translation operation is guaranteed for any Boolean function.

In order that a unique mapping between the two functions exists it is necessary that the functions defined by the set of defining variables $(x_1, \ldots, x_k', \ldots, x_n)$ are linearly independent. If this were not so the expression $F(x_1, \ldots, x_k, \ldots, x_n) \triangleq F'(x_1, \ldots, x_k', \ldots, x_n)$ would imply that the variables $(x_1, \ldots, x_k, \ldots, x_n)$ were not linearly independent , whereas in fact they are. (They represent the minimum number of defining variables necessary to define all points in n-space). A unique mapping of $F(x_1, \ldots, x_k, \ldots, x_n)$ onto $F'(x_1, \ldots, x_k', \ldots, x_n)$ is therefore guaranteed provided that the functions given by the defining variables $(x_1, \ldots, x_k', \ldots, x_n)$ are linearly independent.

Using Galois Field 2 , (GF(2)), theory it is possible to represent the set of defining variables $(x_1, \ldots, x_k', \ldots, x_n)$ in matrix form and establish the linear independence of each member of the set.

GF(2) theory applies to integers in the range (0,1) together with the operation addition modulo 2. ($\oplus$). Because a field is being considered conventional matrix algebra may be employed and the normal criteria of singularity and non-singularity applies to the linear independence of functions.

Under GF(2) the following relationships hold :

1/ Multiplication '.'

$$0.0 = 1.0 = 0.1 = 0$$

$$1.1 = 1$$

2/ Addition '+'

$$0+0 = 1+1 = 0$$

$$0+1 = 1+0 = 1$$

3/ Subtraction is equivalent to addition.

In order that the linear independence of a set of functions may be tested it is necessary to establish that the matrix , in GF(2), describing those functions is non-singular.

Example

A set of defining variables is given by (x_1', x_2, x_3, x_4) , where $x_1' = x_1 \oplus x_2$. Are the functions corresponding to these defining variables linearly independent ?

Expressing the problem in matrix form GF(2) gives

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 \oplus x_2 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

In order that the functions corresponding to the defining variables are linearly independent it is necessary that the above matrix is non-singular. ie. it has a determinant of value 1 . Let this matrix be denoted by $[\Lambda]$.

Expanding the determinant of $[\Lambda]$ by the first column in the usual way gives

$$1. \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

. Re-expansion of this determinant by the first column gives

$$1.\left\{1.\begin{vmatrix}1 & 0\\ 0 & 1\end{vmatrix}\right\} = 1.1.\left\{(1.1) + (0.0)\right\}$$
$$= 1.1.\left\{1 + 0\right\}$$
$$= 1.1.1$$
$$= 1$$

That is Det.$[\Lambda] = 1$, therefore the defining variables are linearly independent.

—oOo—

For the more general case where the variable x_1 is replaced by $x_1 \oplus \left\{x_a \oplus x_b \oplus \ldots \oplus x_h\right\}$, $[\Lambda]$ becomes

$$\begin{bmatrix}
1 & * & * & * & * & . & . & . & . & * & * \\
0 & 1 & 0 & 0 & 0 & . & . & . & . & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & . & . & . & . & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & . & . & . & . & 0 & 0 \\
. & . & . & . & . & . & . & . & . & . & . \\
 & & & & & & & & & & \\
. & . & . & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & . & . & 0 & 1
\end{bmatrix}$$

where $*$ denotes a value of 0 or 1 .

Expanding the determinant of $[\Lambda]$ about the first column gives

$$1.\begin{vmatrix}
1 & 0 & 0 & 0 & . & . & . & . & 0 & 0 \\
0 & 1 & 0 & 0 & . & . & . & . & 0 & 0 \\
0 & 0 & 1 & 0 & . & . & . & . & 0 & 0 \\
. & . & . & . & . & . & . & . & . & . \\
 & & & & & & & & & \\
. & . & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & 0 & 0 & 0 & . & . & 0 & 1
\end{vmatrix}$$

which again give a value of the determinant of $[\Lambda]$ as 1.

The same result is obtained for the general case where x_k is replaced by $x'_k = x_k \oplus \left\{x_a \oplus x_b \oplus \ldots \oplus x_h\right\}$ where the determinant of $[\Lambda]$ is evaluated by expansion about the k th column.

It can be concluded therefore that the operation of spectral translation maps a given function uniquely onto a new function. That is , the minterms of the original function are perturbed in n-space and no information about the original function is lost - it is reconstructable.

The set of defining variables of a function are also termed a Basis and operations of the type considered are often called Basis Transformations. See also reference 29 .

APPENDIX 3

CANONIC SPECTRA OF BOOLEAN FUNCTIONS , $n \leqslant 4$,
UNDER TRANSLATIONAL-EQUIVALENCE.

Fn. No.	0	1	2	3	4	12	13	14	23	24	34	123	124	134	234	1234	ORDER n	Spectral Coeff.
1	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	T
2	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	T
3	14	2	2	2	2	-2	-2	-2	-2	-2	-2	2	2	2	2	-2	4	T
4	2	14	2	2	2	-2	-2	-2	2	2	2	-2	-2	-2	2	-2	4	T
5	0	12	4	4	4	0	0	0	0	0	0	-4	-4	-4	4	0	4	T
6	12	4	4	4	0	-4	-4	0	-4	0	0	4	0	0	0	0	3	T
7	4	12	4	4	0	-4	-4	0	4	0	0	-4	0	0	0	0	3	T
8	10	6	6	2	2	-6	-2	-2	-2	-2	2	2	2	-2	-2	2	4	T
9	6	10	6	2	2	-6	-2	-2	2	2	-2	-2	-2	2	-2	2	4	T
10	2	10	6	6	2	-2	-2	2	2	-2	-2	-6	-2	-2	2	2	4	T
11	0	8	8	8	0	0	0	0	0	0	0	-8	0	0	0	0	3	T
12	8	8	8	0	0	-8	0	0	0	0	0	0	0	0	0	0	2	T
13	8	8	4	4	4	-4	-4	-4	0	0	0	0	0	0	-4	4	4	T
14	4	8	8	4	4	-4	0	0	0	0	-4	-4	-4	0	0	4	4	T
15	0	8	8	4	4	0	4	-4	-4	4	0	-4	-4	0	0	0	4	
16	6	6	6	6	6	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	6	4	T
17	2	6	6	6	6	-6	2	2	2	2	-6	-2	-2	-2	-2	2	4	
18	4	4	4	4	4	4	4	-4	-4	4	-4	-4	-4	4	4	-4	4	

APPENDIX 4.

CANONIC CHARACTERISTIC WEIGHT-THRESHOLD VECTORS, or CHOW PARAMETERS, FOR THRESHOLD FUNCTIONS OF UP TO n = 5.

n(max)	No.	c]	w]
3			
	1	8 0 0 0	1 0 0 0
	2	6 2 2 2	2 1 1 1
	3	4 4 4 0	1 1 1 0
4			
	1	16 0 0 0 0	1 0 0 0 0
	2	14 2 2 2 2	3 1 1 1 1
	3	12 4 4 4 0	2 1 1 1 0
	4	10 6 6 2 2	3 2 2 1 1
	5	8 8 8 0 0	1 1 1 0 0
	6	8 8 4 4 4	2 2 1 1 1
	7	6 6 6 6 6	1 1 1 1 1
5			
	1	32 0 0 0 0 0	1 0 0 0 0 0
	2	30 2 2 2 2 2	4 1 1 1 1 1
	3	28 4 4 4 4 0	3 1 1 1 1 0
	4	26 6 6 6 2 2	5 2 2 2 1 1
	5	24 8 8 4 4 4	4 2 2 1 1 1
	6	24 8 8 8 0 0	2 1 1 1 0 0
	7	22 10 10 6 2 2	5 3 3 2 1 1
	8	22 10 6 6 6 6	3 2 1 1 1 1
	9	20 12 12 4 4 0	3 2 2 1 1 0
	10	20 12 8 8 4 4	4 3 2 2 1 1
	11	20 8 8 8 8 8	2 1 1 1 1 1
	12	18 14 14 2 2 2	4 3 3 1 1 1
	13	18 14 10 6 6 2	5 4 3 2 2 1
	14	18 10 10 10 6 6	3 2 2 2 1 1
	15	16 16 16 0 0 0	1 1 1 0 0 0
	16	16 16 12 4 4 4	3 3 2 1 1 1
	17	16 16 8 8 8 0	2 2 1 1 1 0
	18	16 12 12 8 8 4	4 3 3 2 2 1
	19	14 14 14 6 6 6	2 2 2 1 1 1
	20	14 14 10 10 10 2	3 3 2 2 2 1
	21	12 12 12 12 12 0	1 1 1 1 1 0

APPENDIX 5

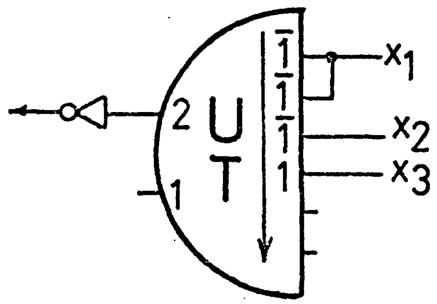
CANONIC SPECTRA OF BOOLEAN FUNCTIONS , $n \leqslant 4$,
UNDER DISJOINT-TRANSLATIONAL-EQUIVALENCE.

Fn. No.																	
1	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	T
2	14	2	2	2	2	-2	-2	-2	-2	-2	-2	2	2	2	2	-2	T
3	12	4	4	4	0	-4	-4	0	-4	0	0	4	0	0	0	0	T
4	10	6	6	2	2	-6	-2	-2	-2	-2	2	2	2	-2	-2	2	T
5	8	8	8	0	0	-8	0	0	0	0	0	0	0	0	0	0	T
6	8	8	4	4	4	-4	-4	-4	0	0	0	0	0	0	-4	4	T
7	6	6	6	6	6	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	6	T
8	4	4	4	4	4	4	4	-4	-4	4	-4	-4	-4	4	4	-4	
	0	1	2	3	4	12	13	14	23	24	34	123	124	134	234	1234	Spectral Coeff.

APPENDIX 6

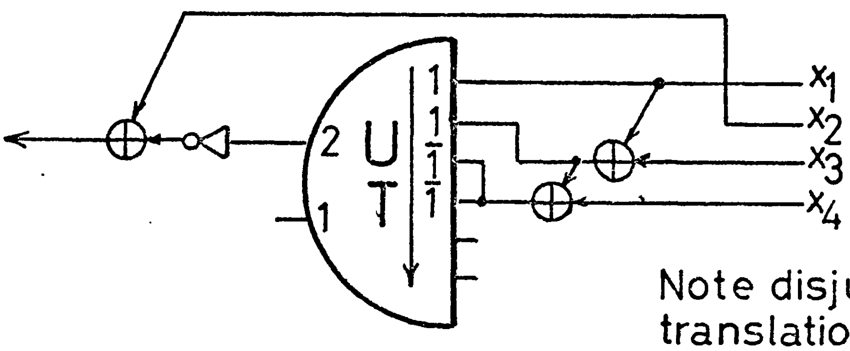
Some Circuits Designed Using the Optimised Universal Threshold Gate[+].

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	0
10	0	0	1	0



Compare with the solution of Fig. 30

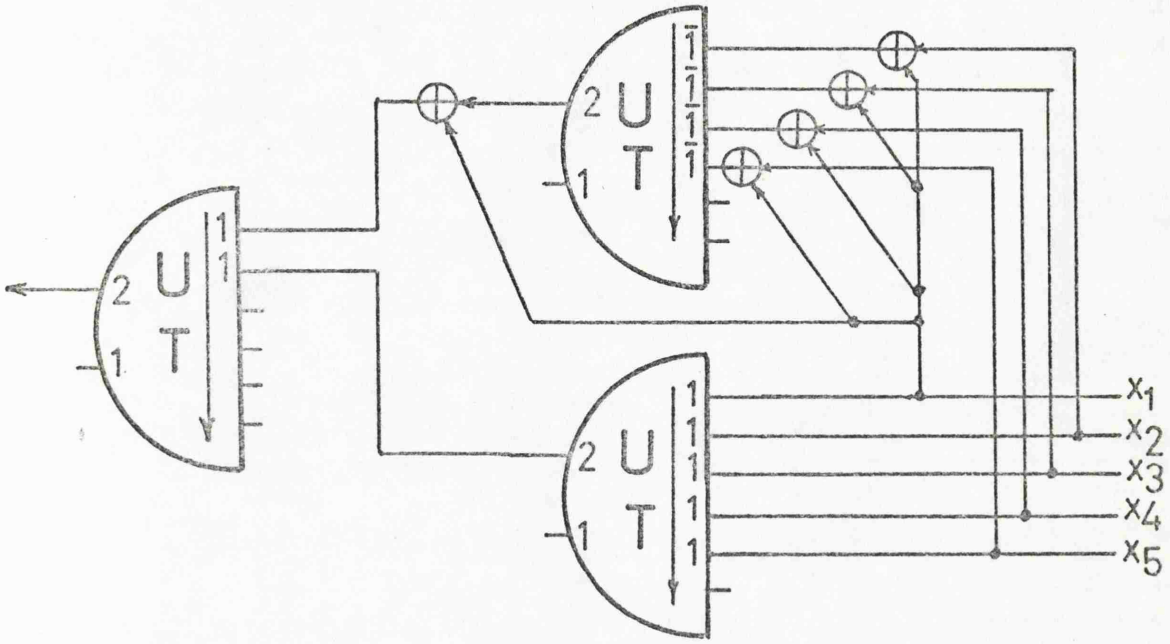
$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	1	1	0
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0



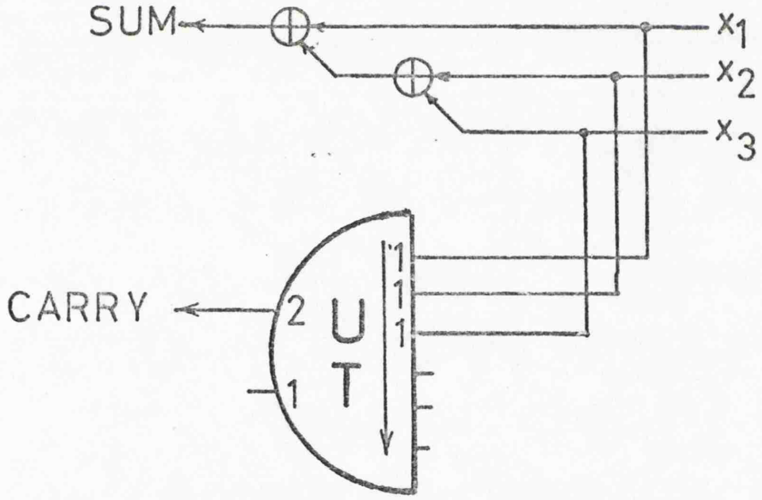
Note disjunctive translation.

Compare with Fig. 31b.

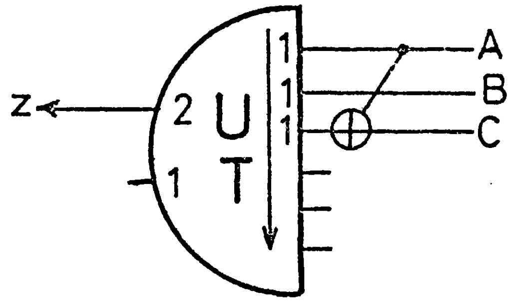
[+](O.U.T.G) with complemented input capability, see Fig. 44b.



2/5 Circuit (Saving of 3 gates & 5 interconnections on Fig.38)



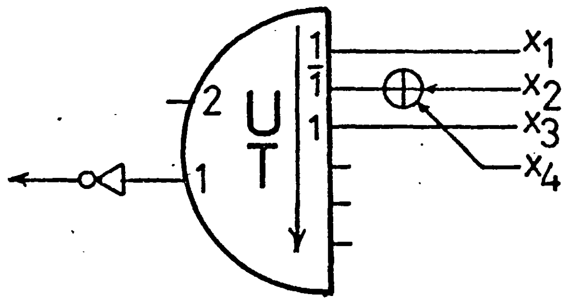
Full Adder



If C=0 z=A
If C=1 z=B

<u>Electronic Switch</u>

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	0	0	0
10	0	0	0	0



Output threshold 1 used

REFERENCES.

1/ Lewin, D. 'Logical Design of Switching Circuits'. Nelson. 1968.

2/ Campeau, J.O. 'The Synthesis and Analysis of Counters in Digital Systems by Boolean Matrices'. Masters Thesis . University of California. 1955.

3/ Campeau, J.O. 'The Synthesis and Analysis of Digital Systems'. I.R.E. Trans.Electronic Computers. Vol. EC6. pp 231-241. 1957.

4/ Flegg, H.G. 'Boolean Algebra and its Applications'. Blackie. 1964.

5/ Edwards, C.R. 'The Logic of Boolean Matrices'. Computer J. Vol. 15 No.3 . pp 247-253 1972.

6/ Hennie, F.C. 'Finite State Models for Logical Machines'. Wiley. 1968.

7/ Minnick, R.C. 'Cellular Arrays for Logic and Storage'. Stanford Research Inst. California. 1966.

8/ Edwards, C.R. 'Some Boolean Matrix Operators'. El. Lett. Vol.8. No.5 .pp 113-115. 1972.

9/ McCluskey, E. 'Minimisation of Boolean Functions'. Bell Syst. Tech. J. Vol.35 pp 1417-1444. 1956.

10/ Quine, W.V. 'The Problem of Simplifying Truth Functions'. Am.Math.Mon. Vol. 59 . pp 521-525.

11/ Edwards, C.R. 'Partitioning of Boolean Functions by Variable Complementation'. El. Lett. Vol.8 No.5 pp 138-140.

12/ Roth, J.P. et al. 'A Computer Program for the Synthesis of Combinational Switching Circuits'. Proc. 2nd Ann. Symp. on Switching Circuit Theory and Design. Pub. A.I.E.E. 1961.

13/ Ashenhurst, R. 'The Decomposition of Switching Circuits'. Harvard Computation Lab. Bell Lab. Report No. BL-1(II). 1952.

14/ Edwards, C.R. 'An Algorithm Applicable to Logic Circuit Synthesis' EL. Lett. Vol.8 No.17. pp 442-444. 1972.

15/ Rademacher, H. 'Einige Sätze über Reihen von allgemeinen Orthogonalfunktionen'. Math. Ann. Vol.87. pp 112-138. 1922.

16/ Walsh, J.L. 'A Closed Set of Normal Orthogonal Functions'. Amer. J. Math. Vol.45. pp 5-24. 1923.

17/ Davies, A.C. 'Some Basic Ideas about Binary Discrete Signals'. Symp. Theory and Application of Walsh Functions. Hatfield Poly.(UK) 1971.

18/ Paley, R.E.A.C. 'A Remarkable Series of Orthogonal Functions(I)'. Proc. Lond. Math. Soc. Vol. 34. pp 241-279. 1931.

19/ Chow, C.K. 'On the Characterisation of Threshold Functions'. I.E.E.E. Proc.Symp. Switching Theory and Logic Design. pp 34-38. 1961

20/ Dertouzos, M.L. 'Threshold Logic : A Synthesis Approach'. Research Monograph No. 32. M.I.T. Press. Mass. 1965.

21/ Ito, T. 'Applications of the Walsh Functions to Pattern Recognition and Switching Theory'. Proc. Symp. Applications of Walsh Functions. Naval Research Lab. U.S.A. 1970.

22/ Hurst, S.L. 'The Application of Chow Parameters and Rademacher-Walsh Matrices in the Synthesis of Binary Functions' Computer J. Vol. 16 No. 2. 1973.

23/ Edwards, C.R. 'The Application of the Rademacher/Walsh Transform to Digital Circuit Synthesis' Symp. Theory and Application of Walsh and Other Non-sinusoidal Functions. Hatfield. Poly. (UK). 1973.

24/ Shanks, J.L. 'Computation of the Fast Walsh-Fourier Transform' I.E.E.E Trans. Electron. Comput. (Short Note). Vol. EC-18. pp 457-459. 1969.

25/ Searle, N.H. 'Walsh Functions and Information Theory'. Symp. Theory and Application of Walsh Functions. Hatfield Poly.(UK).1971.

26/ Golomb, S.W. 'On the Classification of Boolean Functions'. I.R.E. Trans. Circuit Theory. Vol CT-6. pp 176-186. 1959.

27/ Winder, R.O. 'Threshold Functions Through n=7'. Scientific Report No.7 . R.C.A. Labs.Princeton N.J. 1964.

28/ Lewis, P.M. 'Practical Guide to Threshold Logic'. Electron. Design. Vol. 22 pp 66-88 .Oct.1967.

29/ Birkhoff, G. Mac Lane, S. 'A Survey of Modern Algebra'. Third Ed. Macmillan. 1965.

30/ Karp, R.M. et al 'A Computer Program for the Synthesis of Combinational Swithching Circuits'. 2nd Annual Symp. Switching Cct. Theory and Logic Design. Pub. A.I.E.E . 1961.

31/ Wine, R.L. 'Statistics for Scientists and Engineers'. Prentice Hall 1964.

32/ Guild, H.H. 'Some Cellular Logic Arrays for Non-Restoring Binary Division'. Radio and Electron. Eng. Vol 39 . pp 345-348 . 1970

33/ Dean, K.J. 'Design for A Full Multiplier'. Proc.I.E.E. Vol. 115 No.11 pp 1592-1594. 1968.

34/ Edwards, C.R. 'Floating point Cellular-Logic Multiplier with Variable Dynamic Range'. El. Lett. Vol 7 No.25 pp 747-749 . 1971

35/ Frecon, L. Clair, L. 'Operateurs Virgule-Flottante Dynamique-Variable Precision-Multiple'. El. Lett. Vol.8 No.8. pp 191-193 . 1972

PUBLICATIONS. C.R. Edwards. University of Bath.

1/ 'Floating- point cellular-logic multiplier with variable dynamic range ' . Electronic Letters . 16 Dec. 1971
Vol.7 No. 25 . p 747 fol.

2/ 'Partitioning of Boolean function by variable complementation'
Electronic Letters . 9 Mar. 1972
Vol. 8 No. 5 . p 138 fol.

3/ 'Some Boolean matrix operators ' . Electronic Letters .
9 Mar. 1972 . Vol 8 No.5. p 133 fol.

4/ 'An algorithm applicable to logic-circuit synthesis'
Electronic Letters . 24 Aug. 1972.
Vol. 8 No.17. p 442 fol.

5/ 'The logic of Boolean matrices ' . The computer journal (U.K.)
Vol. 15 No. 3 pp247-253.

6/ 'The application of the Rademacher/Walsh transform to digital circuit synthesis '
The Hatfield Symposium on Walsh and other non-sinusoidal functions . Hatfield Polytechnic .Hatfield. England. June 28-29 1973.

7/ ' The synthesis of logic functions by threshold or vertex gates under the Rade macher/Walsh transform '
Under revision for publication. IEEETC.
Copy available as internal report Univ.Bath.

PATENTS.

1/ 'Cellular Logic Arrays ' . U.K. Patent 51122/71. Filed 30 Jan. 1973.

2/ ' Single line communication '. British patent 8047/70.
(Joint Patent)