Cellular bulk transfer system
by Kalyan Kumar Roy

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirements for the degree of DOCTOR OF PHILOSOPHY in Electrical Engineering
Montana State University

Abstract:
In this thesis the results of investigations on a cellular bulk transfer system from the viewpoint of its logical capabilities have been presented. The model adopted for the bulk transfer system consists of an input array, a mapping device, an output array and an output logic. The influence of such factors as flexibility of the mapping device, flexibility of output logic and parallelism of operation has been determined. The main results obtained are: the bulk transfer system can be made logically universal with a proper combination of output logic and maps. In realizing arbitrary logic, a trade-off among the number of mapping operations, number of independent maps and amount of logical flexibility in the output logic is possible. A least upper bound on the number of necessary transposition maps is derived for an output logic consisting of a flexible cellular cascade. The possibility of a set of bulk transfer units operating in parallel has been studied and the functions realizable in this manner have been characterized. An algorithm for test-synthesis of realizable functions has been presented.

CELLULAR BULK TRANSFER SYSTEM

by

KALYAN KUMAR ROY

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Paul E. Schlinch
Head, Major Department

A. Mukhopadhyay
Chairman, Examining Committee

J. Boerng
Graduate Dean

MONTANA STATE UNIVERSITY
Bozeman, Montana

March, 1970

ACKNOWLEDGEMENT

The author wishes to offer grateful thanks to Dr. Amar Mukhopadhyay for many discussions and suggestions during the course of this work. He is also grateful to Professor R.C.Minnick for many helpful suggestions during this period.

The financial support for the graduate study through the award of a research assistantship by the Department of Electrical Engineering, Montana State University, a teaching assistantship by the Department of Computer Science, University of Iowa and through National Science Foundation Grant nos. GJ 158 and GJ 723 is thankfully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In this thesis the results of investigations on a cellular bulk transfer system from the viewpoint of its logical capabilities have been presented. The model adopted for the bulk transfer system consists of an input array, a mapping device, an output array and an output logic. The influence of such factors as flexibility of the mapping device, flexibility of output logic and parallelism of operation has been determined. The main results obtained are: the bulk transfer system can be made logically universal with a proper combination of output logic and maps. In realizing arbitrary logic, a trade-off among the number of mapping operations, number of independent maps and amount of logical flexibility in the output logic is possible. A least upper bound on the number of necessary transposition maps is derived for an output logic consisting of a flexible cellular cascade. The possibility of a set of bulk transfer units operating in parallel has been studied and the functions realizable in this manner have been characterized. An algorithm for test-synthesis of realizable functions has been presented.

Chapter 1

INTRODUCTION

1.1 Introduction

A cellular array is some geometrical arrangement of cells in one, two or three dimensions. Each cell in a cellular array has some logical property and it may also have some storage capability. The cells of an array have a uniform interconnection structure. Because of this, the logic designer is faced with a new kind of constraint in realizing arbitrary logic functions. With existing techniques in cellular logic[11] it is necessary to use either a complicated interconnection pattern among cells or a very large number of cells to realize arbitrary logic.

The idea of a bulk transfer of data originated from the problems of realizing arbitrary logic functions with cellular arrays. With a view to avoiding the complicated interconnection pattern, it has been proposed to transfer the data from one cellular array to another by some kind of transferring device. The advantage in doing this is that necessary interconnections can be realized by suitably transferring the inputs in the second array. Pursuing this line, a cellular bulk transfer system can be conceived which consists of two cellular arrays which may be referred to as input and output arrays. Each of these arrays is capable of containing data and possibly performing logic on it. Between the two arrays there is a data-transfer device which may transfer data,

accompanied by some kind of transformation. An example of a simple type of transformation is a permutation of the variables on the array. If necessary, the device may have the capability for more complex transformations This device may be given the general name 'mapping device'. The mapping may be applied in reverse direction and can be iterated a limited number of times. Data may be logically processed using the built-in logic in the cellular arrays, maps in the mapping device and if required, by a separate logic device.

A study of the characteristics of this system reveals that logical universality can be achieved by a suitable combination of map and logic. This leads to many interesting questions regarding its efficiency, effect of variation in structure and construction, capability, and practicality as a computing system and so on. The model of the bulk transfer system is related to the perceptron system studied by Minsky and Papert[13] but the major interest in the system is based on a realization that it may have an important place in future digital systems because of a possibility that improvement in the computing power may be achieved through a mode of operation in which the advantages of increased parallelism of operations, greater homogeneity of the hardware structure, intermixture of storage,

arithmetic and control operations have been combined.

The generality of the system and its potential capabilities may be visualized by considering the basic schemes of the parallel processors suggested recently and attempting to mirror their functions into the bulk transfer system. Most of these machines have utilized the basic concepts of two distinct parallel processor organization proposed by Unger[17] and Holland[7]. These machines will be briefly discussed in order to bring out functional similarities between these and the bulk transfer system.

1.2 The Bulk Transfer System in Relation to Some Parallel Processors

Unger Machine

In 1958 Unger described a stored program computer oriented toward spatial problems. The particular problem illustrated with this machine was pattern detection. The computer consists of a master control and a rectangular array of logical modules each of which can communicate with its nearest neighbors (Figure 1.2.1). The master control contains a clock, decoding circuits and a random access memory for storing instructions. It reads out instructions from memory, decodes them and sends out appropriate commands which go simultaneously to all the modules. Each module contains a one-bit

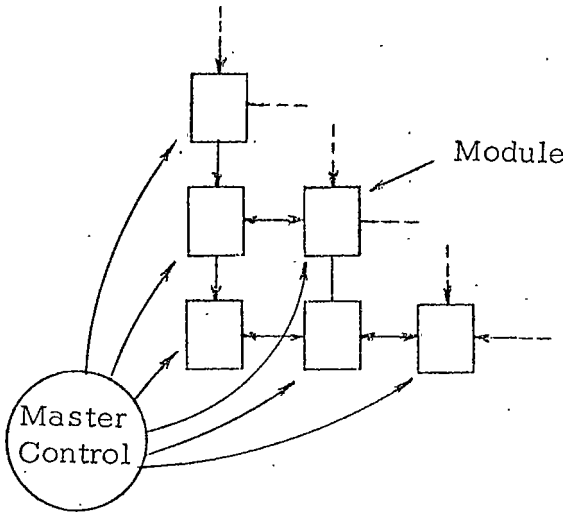


Figure 1.2.1 The Unger Machine

accumulator, some storage and associated logic and works in parallel with the remaining modules. With the use of an elementary instruction set Unger has described programs to detect certain local and global features of patterns on the two dimensional field.

To reflect these features of the Unger machine in the bulk transfer system, let us regard the field consisting of the modules as the input array and assume that the resulting configuration of the field on execution of one instruction will be on the output array. The instruction can be described by a proper combination of mapping with logic. Thus, an iterated procedure of mapping with logic may constitute a program for the determination of some particular feature of a pattern. It appears that the bulk transfer system can be potentially more powerful in some

respects than the Unger machine in view of the fact that suitable choice of maps in the mapping device may enable the bulk transfer system to process a larger part of the field than the group of immediate neighbors. Furthermore, some operations can be conveniently executed by a simple mapping where it may take several instructions in the Unger machine to do the same. On the other hand, there may be certain features in the Unger machine which may be difficult to implement by mapping in the bulk transfer system, but can be incorporated through the built-in logic of the input and output arrays.

Holland Machine

The Holland machine consists of a two-dimensional array of modules. Each module is a small general purpose computer and can be active or inactive at a given time. When active, a module treats the contents of its storage register as an instruction. After determining the location of the operand, a path is built to access the data. The instruction in the module is then executed and the active status is transferred to one of the four nearest neighboring modules in the array. Instructions can be arranged spatially throughout the array of modules with an arbitrary number of these executed at the same time.

Reflecting on the B.T. system, it appears that, though any data transformation possible in the Holland machine can be done by performing suitable mapping, a modification of the structure of the mapping device may be more suited to perform the kind of parallel computation that the Holland machine is able to do. Let the input array be the proper configuration containing data and instructions. The mapping device may be divided into many separate elements, each having the capability of scanning a limited area of any region in the input array. The execution of an instruction may consist of a transformation of data, first by mapping and then by the built-in logic of the arrays. If the operand of an instruction belongs to an area different from that scanned by a mapping element, a sequence of mapping operations may be undertaken to bring the data into the proper area, thus simulating the path-building and data-access procedure in the Holland machine. There could be other variations of this simulation procedure including a global mapping.

Two-dimensional Computing Technique

The bulk transfer system is structurally oriented toward parallel computation in the same manner as the two-dimensional iterative network computing technique of Hawkins and Munsey[7]. The computing machine devised by these authors consists of two data planes,

called the input plane and resultant plane (Figure 1.2.2). There is an intervening plane called the mapping mask. Data on the input plane can be processed through the use of the mask and projected onto the resultant plane. Using optical techniques and linear threshold logic to process spatially distributed data on the input plane, the authors have shown that the system is capable of recognizing certain tiny objects against the background of other large objects.

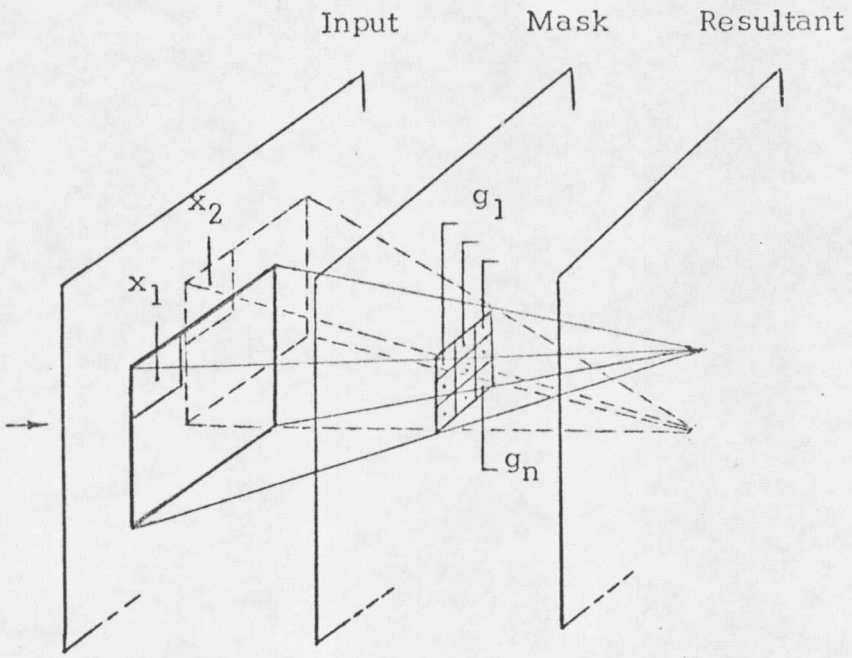


Figure 1.2.2 Optical Summation System
(x's are input variables, g's are transmittances)

From the above discussions it should be apparent that given the proper technological background, the bulk transfer system has the potentialities of becoming an improved, economic and versatile data-processing device. The technological advance in the field of batch fabrication technique has made it possible to produce reliable and economical arrays of logic cells on a mass scale [11]. Current research [1] on the use of optical techniques for data transfer on a large scale points to a direction in which the practical realization of a bulk transfer system may seem feasible.

With these aspects in view, a study on the logical capabilities of the bulk transfer system has been proposed in this thesis. The contents and organization of the thesis is given in the following section.

1.3 Organization of the Remaining Chapters

Chapter 2 discusses the logical capability of a simple bulk transfer system. The results obtained in this chapter form the basis of further development on this topic in Chapter 3. In this chapter the effect of limited flexibility in the output logic is studied and results on bulk transfer with cellular cascades are reported.

Chapter 4 deals with the characterization of realizable functions using parallel bulk transfer technique. It contains an algorithm for the test-synthesis of functions using parallel bulk transfer on the assumption that the partioning of the input array is fixed. In Chapter 5 a similar algorithm on the assumption of flexibility in the paritioning of the input array has been developed.

Chapter 6 gives a summary of the foregoing chapters followed by a discussion about the scope of further research work in the area.

Chapter 2

LOGICAL CAPABILITY OF A BULK TRANSFER SYSTEM

2.1 A Bulk Transfer System

An idea of the possible function of a bulk transfer system has been given in the previous chapter. Such a system will be described in this chapter and then investigations into its logical capabilities will be made. A block diagram of the system is shown in Figure 2.1.1. The block called Input Array may be conceived as a rectangular plane divided into many rectangular cells. Each cell is a storage device which can take on one of the two input values, 1 or 0. Corresponding to each input cell there is a binary variable and an assignment of input values to all the cells represents one of 2^n possible input configurations, where n is the number of cells in the input array.

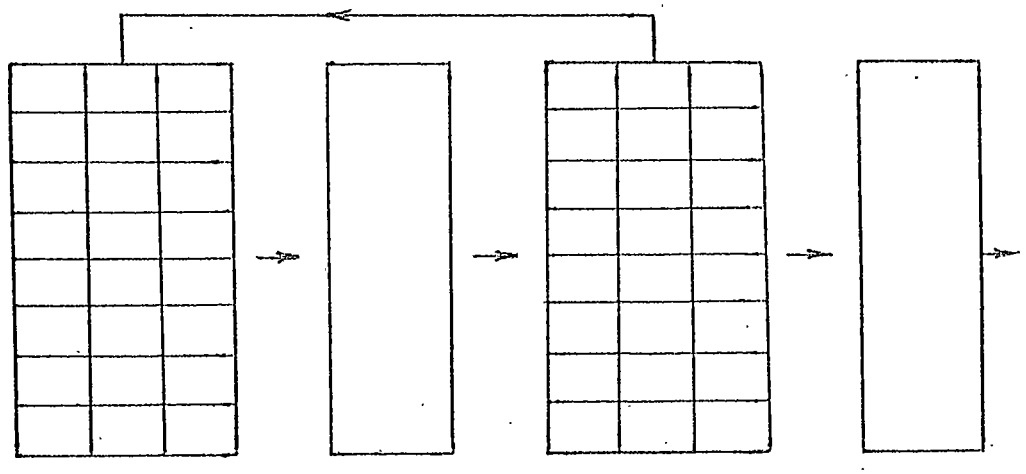


Figure 2.1.1 A Bulk Transfer System

A second component of the bulk transfer system is the block called the Output Array: it is a plane identical to the input array. The contents of the input cells are mapped (transferred) onto the cells of the output array. For example, a stored bit in a cell (i,j) in the input array may be mapped into a cell (m,k) in the output array.

The third component is called the Mapping Device: it implements the mapping operation between the input and output arrays. The mapping may include logic (Figure 2.1.2) or may not. A mapping is called logic-free if the value of an input bit is not changed during mapping. For the present discussion, such a mapping is always one-to-one; that is, each output cell gets a signal from at most one

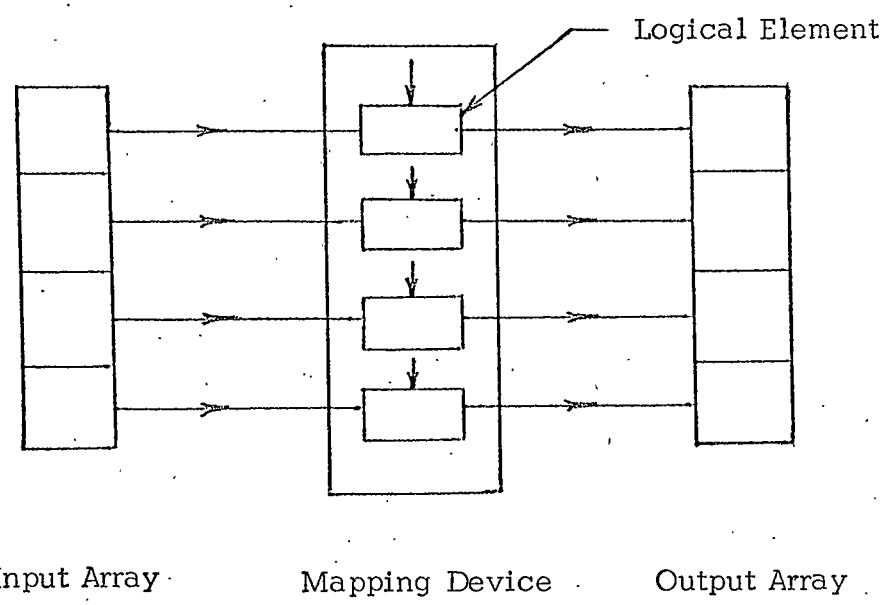


Figure 2.1.2 Mapping Device with Logic

input cell. (Note that all mappings have been assumed to be onto, that is, an output cell gets a signal from at least one input cell.) A logical mapping is one that involves transformation of the input bits during mapping (for example, complementation). Besides being one-to-one, a logical mapping may be many-to-one, where a number of input bits combine with some logic to produce a binary signal which is then mapped into an output cell.

After one mapping, the output array can be identically mapped back into the input array for a second mapping from input array to output array. Assume that the previous assignment of switching values to the cells in an array is automatically cleared when a new mapping to the array is performed. Also, assume that a mapping from the input array to the output array can be repeated any number of times by identically mapping back from the output array to the input array after each mapping from the input array to the output array. Henceforth, by "mapping" the mapping from input array to output array will be meant, unless otherwise specified, because the reverse mapping is assumed always to be the identity mapping.

The mapping device is capable of producing a limited number of independent mappings, none of which can be produced by any composition of other mappings that the device implements.

The final component of the bulk transfer system is called the Output Logic: it is a device that performs a logical function on the output array to produce an output value. Assume that during composite mapping this device does not operate until the end of composition. The logic of the device may be flexible so that different functions of its inputs can be obtained.

Note that an assignment of values to the input array of n cells represents a minterm of n variables which is converted into another minterm in the output array after mapping. Consider the set M of all minterms of n variables. Clearly, the mapping device can produce a mapping from M into M. Let a minterm m_1 after a mapping α be changed to another minterm represented by $m_1\alpha$. The same minterm when subjected to the map α twice, is changed to a minterm represented as $m_1\alpha^2$. If $m_1\alpha^2$ is different from $m_1\alpha$ and the values of the output function for the two minterms are different, then evidently, a new function (i.e., a function different from that obtainable after single mapping) is generated at the output by one repetition of the α -map.

The bulk transfer system may be looked upon as a device for producing different logical functions of a set of input variables,

using a set of maps ($\alpha_1, \alpha_2, \ldots \alpha_k$) and an output logic device. We shall study the logical capabilities of the system under different types of mapping and output logic. In this connection, some of the basic ideas and terminology of Group Theory[3] will often be used.

2.2 Logical Capability of the Bulk Transfer System

Permutation Maps

Let the set of 2^n minterms be arranged in some order as $(m_1, m_2, \ldots m_{2^n})$. If it were possible to produce all possible permutations of the set of minterms with the help of the limited number of independent maps of the bulk transfer (B.T.) system, then, with an output logic device of very limited flexibility, an extremely large number of functions could be generated, because, by permuting the set of minterms a different function could be obtained at the output without changing the output function of the output device. So the attempt in the following is directed toward this goal. Theorem 2.2.1 is a result of the foregoing discussion, and so is given without proof.

Theorem 2.2.1: A one-to-one map of the input array in a bulk transfer system produces a permutation of the set of minterms.

A logic-free mapping of the input array is effectively a geometrical repositioning of the input bits. This type of mapping has limitations

in the matter of producing arbitrary permutations of the set of minterms as the following theorem will show:

Theorem 2.2.2: With all possible logic-free one-to-one mappings of the input array, it is not possible to produce all permutations of the set of minterms.

Proof: A logic-free mapping cannot map a minterm of index number i (number of 1's in the minterm) into a minterm of index number j, where $i \neq j$. Therefore, such a mapping cannot produce all possible permutations. Q.E.D.

The types and the total number of permutations that all logic-free one-to-one maps produce will be given under Theorem 2.2.5. Suppose the set of minterms is identically mapped and the function produced by the output logic (fixed) is f. Can a different function be produced at the output by resorting to a different permutation of the set of minterms? Consider a permutation of the set of minterms where two minterms m_i and m_j are mapped one into the other while the rest are mapped identically. If the output logic is such that both m_i and m_j are true or both false, then, with this permutation, no new function can be produced. The condition for producing a new function is given in the following theorem.

Theorem 2.2.3: Let f be the function produced with identity mapping by a bulk transfer system having a fixed output logic. Then with any mapping of the input array which produces a permutation of the set of minterms, it is possible to produce a new function at the output if and only if the corresponding permutation of the set of minterms maps at least one true (false) minterm into a false (true) minterm.

Proof: The proof follows easily from earlier discussions and so is omitted.

Theorem 2.2.4 gives the maximum number of functions realizable by a bulk transfer system with a fixed map and a fixed output logic.

Theorem 2.2.4: A bulk transfer system with a single one-to-one map and with fixed output logic can realize at most k functions where k is the order of the corresponding permutation of the set of minterms.

Proof: Since with repeated applications of the map, only k different permutations of the set of minterms are produced, hence, if every permutation generated a new function, at most k functions could be produced. Q.E.D.

According to this theorem, out of 256 functions of three variables, at most 15 functions may be realized by a bulk transfer system with a single map and output logic. If there is no restriction about the number of logic-free maps that the mapping device can implement, then the maximum number of functions that a bulk transfer system with fixed output logic can realize, is given by the following theorem:

Theorem 2.2.5: A bulk transfer system with a fixed output logic and capable of producing all logic-free one-to-one maps can realize at most

$$\prod_{i=0}^{n} \binom{\binom{n}{i}}{a_i}$$ functions

where n = number of input cells,

i = index number of a minterm,

a_i = number of true minterms in the output logic function that have index number i, where $0 \leq a_i \leq \binom{n}{i}$

Proof: Let the set of 2^n minterms be divided into subsets such that a subset G_i contains all minterms of index number i. There will be (n+1) such subsets, i ranging from 0 to n. A subset G_i will contain $\binom{n}{i}$ members. If the output logic is such that only a_i minterms of

these $\binom{n}{i}$ minterms are true, then the number of possible combinations of these a_i minterms is $\binom{\binom{n}{i}}{a_i}$. Now, in logic-free mapping, the permutation of the set of minterms is restricted so that a minterm of index number i is mapped to another minterm of same index number. If we assume all possible permutations within the subset of minterms of same index number then the number of different functions that can be generated by permutations of the subset G_i is $\binom{\binom{n}{i}}{a_i}$. Now each of these ways of generating a function can be linked with each way of generating a function by the members of a different subset G_j. So the maximum number of functions possible is $\prod_{i=0}^{n} \binom{\binom{n}{i}}{a_i}$. Q.E.D.

It has been seen that the set of all logic-free one-to-one maps cannot produce all permutations of the set of minterms. With a set of logical maps, however, it is possible to produce all permutations. Given all possible permutations of M, it is still not possible to realize all functions of n variables unless the output logic is sufficiently flexible. The flexibility of the output logic is an essential factor in

the realization of arbitrary logic with a bulk transfer system capable of producing only permutation maps. The following theorem states this:

Theorem 2.2.6: In order to produce arbitrary functions of input variables, a bulk transfer system having the means to produce all permutations of the set of minterms must have an output logic of sufficient flexibility so that at least $(2^n + 1)$ different functions can be produced by the output logic device.

Proof: Let f_r be the function produced by the fixed output logic device of a B. T. system under identity map where r is number of true minterms. With all possible permutations of M, the total number of functions realized is $\binom{2^n}{r}$. With the fixed output logic and a set of maps to produce all permutations of M, it is not possible to change r, the number of true minterms.

So in order to realize arbitrary functions, the output logic must be flexible and capable of producing each function f_i, $i = 0,1,\ldots 2^n$. These functions are listed below:

(1) f_0 = false for all minterms

(2) f_1 = true for only one minterm

(3) f_2 = true for only two minterms

. . . . . .

$(2^r + 1)$ f_2r = true for only 2^r minterms

. . . . . .

$(2^n + 1)$ f_2n = true for all minterms

The total number of distinct functions that can be realized by the B.T. system in this case is $\binom{2^n}{0} + \binom{2^n}{1} + .. + \binom{2^n}{r} + .. + \binom{2^n}{n} = 2^{2^n}$ Q.E.D.

According to this theorem, the number of functions required of the output logic has been reduced to $(2^n + 1)$ from 2^{2^n} for systems without using bulk transfer, a value proportional to the logarithm of the original number. We can further improve upon this and drastically reduce the number of required functions of the output logic with the use of a special map to be discussed now.

Many-to-one Map

The preceding discussion was concerned with permutations of the set of minterms and, therefore, involved only those mappings of the input array that produced permutations. Consider the many-to-one logical maps that map the set of minterms onto one of its proper subsets. In a special case, a sufficient number of repetitions of such

a map may result in a situation in which all the minterms will be mapped onto a single minterm. Let $\emptyset$ be a mapping such that a minterm m_i is mapped into m_{i+1} except the last minterm m_{2^n} which is mapped into itself. The mapping $\emptyset^k$, $k \leq (2^n-1)$ (i.e., $\emptyset$ is repeated k-times) maps (k+1) minterms onto m_{2^n}.

If the mapping device of the bulk transfer system is capable of producing $\emptyset$ along with arbitrary permutation, and the output logic is fixed and produces only the function f_1 where f_1 = true, only when m_{2^n} is true, then every function listed in the proof of Theorem 2.2.6 except the zero function can be produced with the use of a suitable number of repetitions of the $\emptyset$-map. For example, $\emptyset$ applied thrice will generate the function f_4 (the function is true for four minterms). Therefore, the required flexibility of the output logic comes down to two functions:

1) f_0 (zero function)

2) f_1 (function is true for only one minterm, m_{2^n}).

It has been seen that to produce all possible permutations of the set of minterms, logical maps must be used. The following lemma gives the minimum number of independent logical maps necessary to achieve this.

Lemma 2.2.1: The two permutations (m_1, m_2), $(m_1, m_2, \ldots m_i, \ldots m_{2^n})$ can generate all permutations of the set of minterms.

Proof: It is a well-known result in Group Theory. Texts on Group Theory[6] may be seen for proof.

Incorporating the preceding results, we have:

Theorem 2.2.7: A bulk transfer system with a mapping device capable of producing two permutations (m_1, m_2), $(m_1, m_2, \ldots m_i, \ldots m_{2^n})$ and the map $\emptyset$, and a flexible output logic capable of producing two functions, f_0 and f_1, can realize any function of n variables.

Proof: The proof follows from preceding theorems and remarks.

Note that the flexibility of the output logic is not necessary here; the output logic might be made fixed to produce only the function f_1. In this case, another $\emptyset$ map, mapping each minterm onto a false minterm is necessary for realizing the zero function.

2.3 A Simple Design of the Bulk Transfer System

A design for a bulk transfer system capable of arbitrary logic realization will be given here. The first step is to arrange the set of minterms in some order. The following order (Table 2.3) is

convenient for the cascaded circuit we use here. For convenience, the number of variables is confined to three.

Table 2.3 Three-variable Minterms

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
x_1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1

The design depends on the chosen order of minterms. Hence the sequence $\{ m_1, m_2, \ldots m_{2^n} \}$ is always with respect to a particular system. This sequence is referred to as minterm sequence.

Figure 2.3.1 shows the circuit for generating the permutation cycle $(m_1, m_2, \ldots m_{2^n})$. Each logical unit is a 2-input, 2-output device. The logic used for computing the next state of a variable x_n, designated by x'_n is given by

$$x'_1 = \bar{x}_1; \quad x'_2 = (x'_1 \ominus x_2)$$

For $n \geq 3$, $x'_n = \overline{x'}_{n-1} (x_{n-1} \oplus x_n) + x'_{n-1} x_n$

The circuit to generate $\emptyset$ uses some additional logic with the previous circuit. Figure 2.3.2 shows the circuit. The logic equations are:

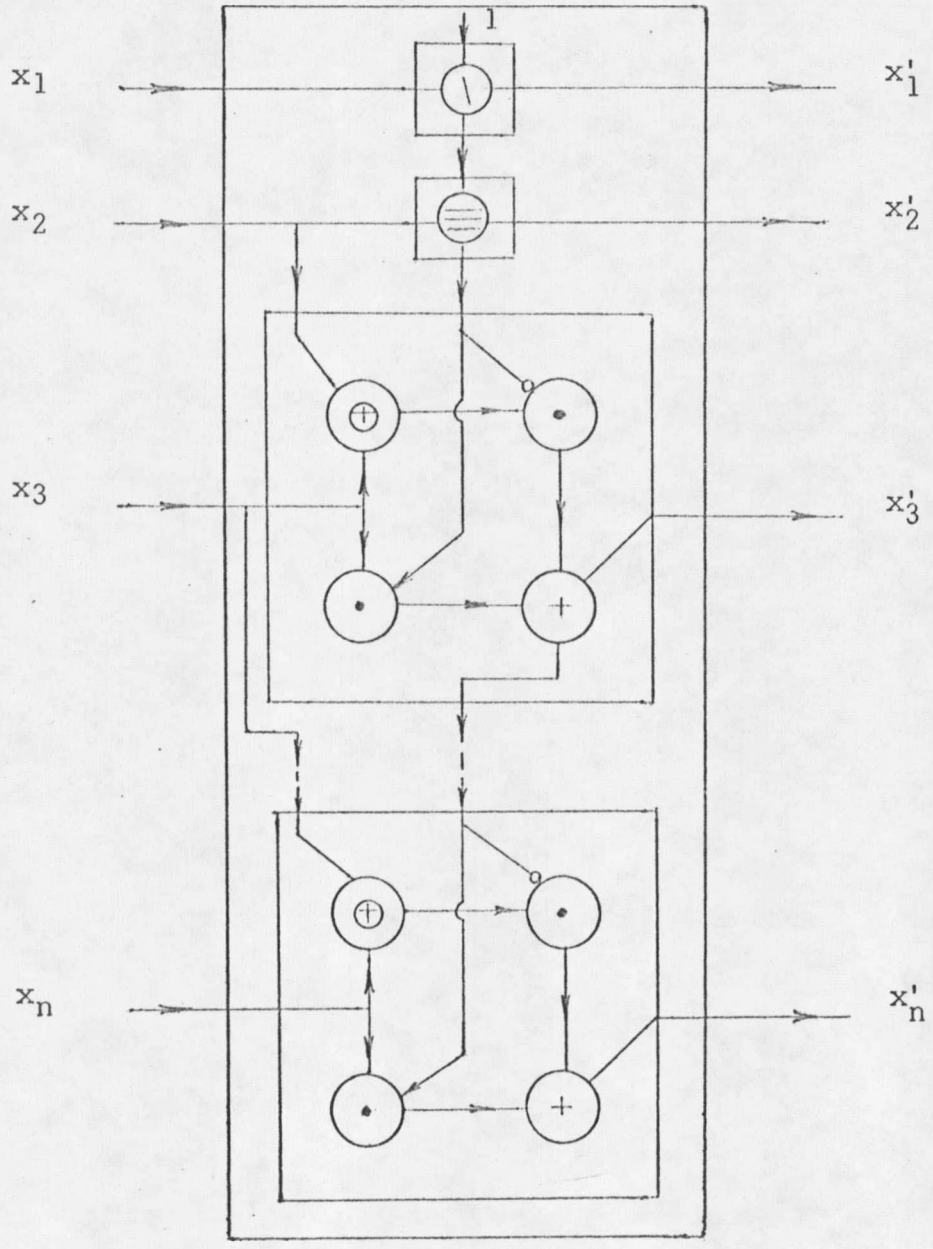


Figure 2.3.1 The Circuit for Generating the Permutation $(m_1, m_2, \ldots m_i, \ldots m_{2^n})$

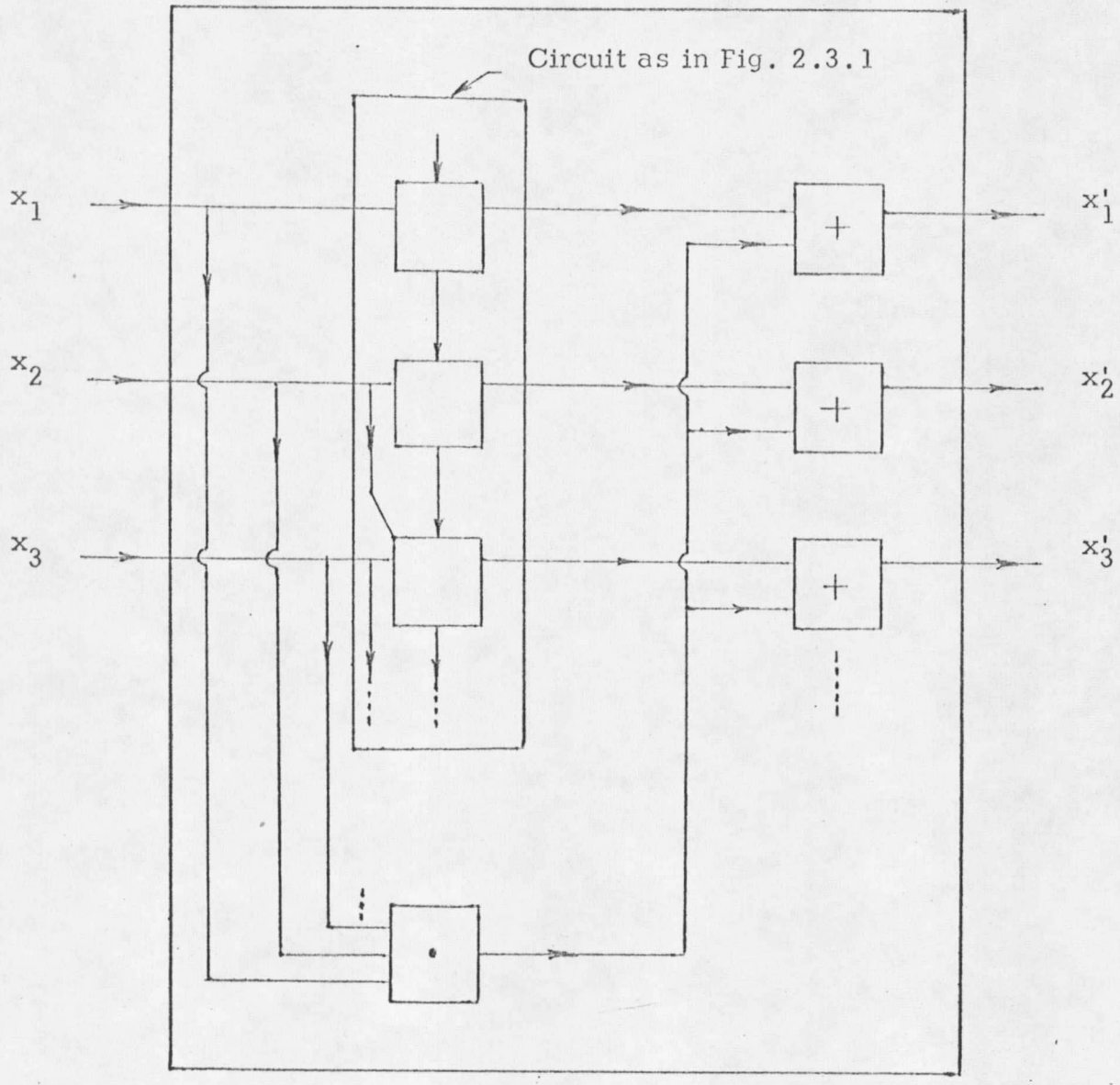


Figure 2.3.2 The Circuit for Generating the Map $\emptyset$

$x'_1 = \bar{x}_1 + x_1x_2..x_i.x_n$; $x'_2 = (x'_1 \circledcirc x_2) + x_1x_2..x_i..x_n$.

For $n \geq 3$; $x'_n = \bar{x}'_{n-1}\,(x_{n-1} \oplus x_n) + x'_{n-1}x_n + x_1x_2..x_i..x_n$

The derivation of these equations are given in Appendix A.

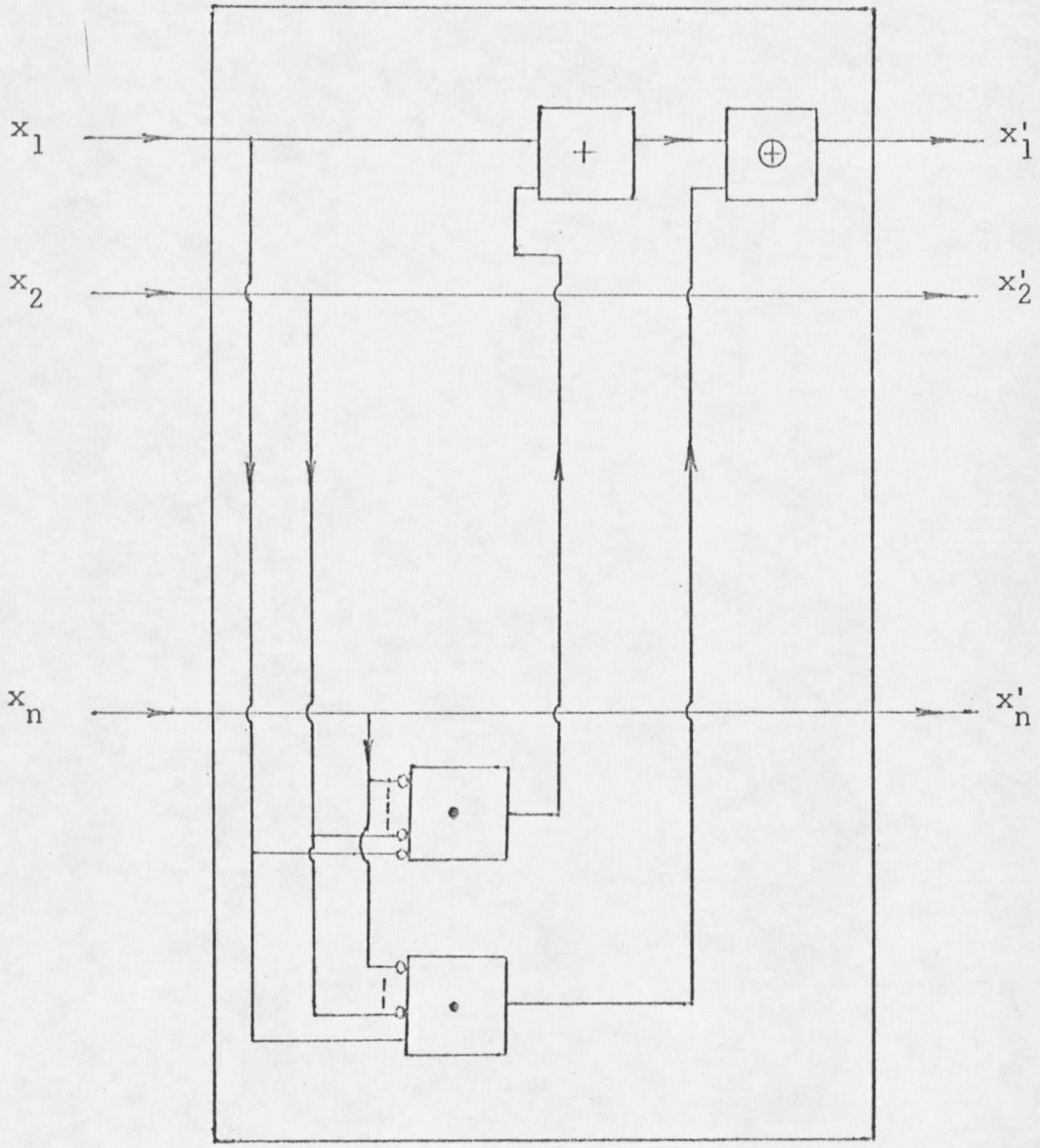
The circuit for generating the permutation (m_1, m_2) is given in Figure 2.3.3. The output logic is built up of a cascade of AND gates (2-input, one output) except the last gate which is flexible and can produce the zero function besides the AND operation.

2.4 Determination of the Necessary Mapping Operations

In order to realize an arbitrary switching function, the mapping operations on the input that will be necessary to generate the function must be known. The following is a systematic procedure for determining the necessary maps. It is assumed here that the function is available in truth table form:

Step 1: Note the number of true minterms. If it is the zero function, then set the output logic to produce the zero function (f_0) and do no more. In this case, any mapping of input array will produce the zero function.

Step 2: If the number of true minterms is k ($\neq 0$), then the map $\phi^{(k-1)}$ (application of ϕ (k-1)-times, in sequence) is to be employed



$x'_1 = (x_1 \oplus x_1\overline{x}_2 \ldots \overline{x}_i \ldots \overline{x}_n) + \overline{x}_1\overline{x}_2 \ldots \overline{x}_i \ldots \overline{x}_n$. For $n > 1$, $x'_n = x_n$

Figure 2.3.3 The Circuit for Generating the Permutation (m_1, m_2)
(o indicates complemented input).

at the end of all other mapping operations.

Step 3: Note the sequence of minterms which has been used in designing the system. List the truth values of the function to be realized against each minterm in the sequence. With the help of the available maps, the permutation that will map the subset of k true minterms into the last k minterms of the minterm sequence, will have to be produced. This permutation can be decomposed into available maps as follows: Note how many of the minterms in the truncated minterm sequence $\left\{ m_{2^n-k+1}, m_{2^n-k+2}, \ldots, m_{2^n} \right\}$ are true.

a. If all of them are true, then apply the $\phi^{(k-1)}$ map of step 2 and then go on to Step 4.

b. If some of the minterms in the truncated sequence are not true, check if the subset of k true minterms form a consecutive sequence within the minterm sequence assuming the last minterm to be adjacent to the first. If so, then use the map for cyclic permutation $(m_1, m_2, \ldots m_{2^n})$; the number of times it should be employed is equal to the distance (measured by the number of minterms in between) between the last minterm in the

minterm sequence and the last true minterm down the block of true minterms.

If the subset of true minterms is interspersed with false minterms in the sequence, then use the map for cyclic permutation a number of times so that a maximum number of true minterms are mapped into the lower positions of the truncated sequence. For each of the remaining true minterms, transposition maps such as (m_i, m_j) can be used, where m_j is a false minterm situated within the truncated sequence $\{ m_{2^{n-k+1}} \cdots m_{2^n} \}$ and m_i is a true minterm outside of this truncated sequence. If m_i, m_j are adjacent, then (m_i, m_j) can be decomposed [6] as

$$(m_i, m_j) = (m_1, m_2, \ldots m_{2^n})^{2^n+1-i} (m_1, m_2) (m_1, m_2, \ldots m_{2^n})^{i-1} \quad \ldots (A)$$

If m_i, m_j are not adjacent, then (m_i, m_j) can be decomposed into a number of other transpositions, each one of them being expressible in form (A). For instance, (m_3, m_5) can be decomposed as $(m_3, m_5) = (m_3, m_4) (m_4, m_5) (m_3, m_4)$. Each component of the righthand expression can now be expressed in form (A).

Step 4. Set the output logic device to produce f_1 in all cases except when the zero function has to be produced (cf. Step 1).

Example 2.4: The following function given in truth table form is to be realized.

x_1	x_2	x_3	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The ordering given in Table 2.3 is used here.

Minterm	F
m_1	0
m_2	0
m_3	1
m_4	0
m_5	1
m_6	1
m_7	1
m_8	1

Step 1: The number of true minterms = 5.

Step 2: $\emptyset^4$ is to be used after all other maps.

Step 3: (m_3, m_4) is to be produced.

$$(m_3, m_4) = (m_1, m_2, \ldots m_8)^6 (m_1, m_2)(m_1 m_2, \ldots m_8)^2$$

Step 4: The output logic is to be set to produce f_1.

Note that a total of 13 mapping operations are needed to produce the function. Such a high number of mapping operations are necessary because only two maps are available in the machine to generate all possible permutations. Suppose, the maps $(m_1, m_2), (m_2, m_3), \ldots, (m_{r-1}, m_r), \ldots, (m_{2^n-1}, m_{2^n})$ are available in hardware. It is known from Group Theory that with this set of transposition maps, all possible permutations can be generated. Then, in the above case, nine mapping operations to produce (m_3, m_4) can be replaced by a single map. Further, the $\emptyset$ maps can be completely eliminated if the output logic is sufficiently flexible so that the (2^n+1) functions, listed in the proof of Theorem 2.2.6 are generated by it. Thus there is a way of reducing the number of mapping operations by increasing the mapping hardware and the flexibility of the output logic.

Note that mapping of the input is necessary because the function to be realized is not available in the output logic. Therefore, the extent to which the output logic is flexible determines in how many of the cases mapping has to be employed, the latter number being in inverse proportion to the former. Since the flexibility of the output logic is limited, some maps have to be used. The larger the number of maps available in hardware, the lesser the number of mapping operations to be needed, in general, for the realization of a switching function and, therefore, faster is the whole operation. Thus the number of maps that should be available in hardware must be determined by speed requirements and cost. A trade-off between flexibility of the output logic and additional maps is also possible and the proper balance should be arrived at on the same basis.

Chapter 3

BULK TRANSFER WITH CELLULAR CASCADES

3.1 Transposition Maps

It has been mentioned in Chapter 2 that the set of transposition maps $\{ (m_i, m_{i+1}) \mid i=1,2,\ldots,2^n-1 \}$ are sufficient to produce arbitrary permutation of the set of minterms. Similar set of sufficient maps can be $\{ (m_1,m_i) \mid i = 2,3,\ldots,2^n \}$. If the mapping device of a B.T. system is assumed to be capable of directly implementing any transposition (m_i,m_j), a question may be asked: What is the maximum number of transpositions necessary to realize a function, given that the output logic is of known flexibility? The total number of different possible transpositions is $\binom{2^n}{2}$. If the output logic is assumed to be flexible enough to produce just the (2^n+1) functions as obtained in Theorem 2.2.6, then the maximum number of transpositions that may be necessary to realize an arbitrary function is 2^{n-1}. One way in which the output logic can be made flexible is by using a Maitra cascade[12]. In the next section a measure of the flexibility of Maitra cascade in relation to bulk transfer system will be obtained.

3.2 Output Logic with Maitra Cascade

Maitra[9] has introduced a type of one-dimensional graph format for representing switching functions and with its help, characterized the cascade-realizable functions. These ideas will be used to prove a theorem bounding the number of transposition maps

required for realizing a logic function in a bulk transfer system with a Maitra cascade in the output logic. The one-dimensional graph format is shown in Figure 3.2.1 for n = 3. In constructing the graph, at first an ordering of the variables is assumed (as in the above case: x_3, x_2, x_1). A horizontal line is divided into two equal parts in which the last variable x_3 is equal to 0 over the right-half portion and is equal to 1 over the left-half portion. The region over which $x_3 = 0$ is again divided into two parts, of which, the right-half portion has the next variable, x_2 equal to 0 and the left-half portion, x_2 equal to 1. The values of x_2 over the original left-half portion, where $x_3 = 1$, are a mirror image of those in the right half. In the same manner, the rightmost region over which $x_2 = 0$ is divided into two parts, with the right and left portions having $x_1 = 0$ and $x_1 = 1$ respectively. The values of x_1, in the region next to the left, having

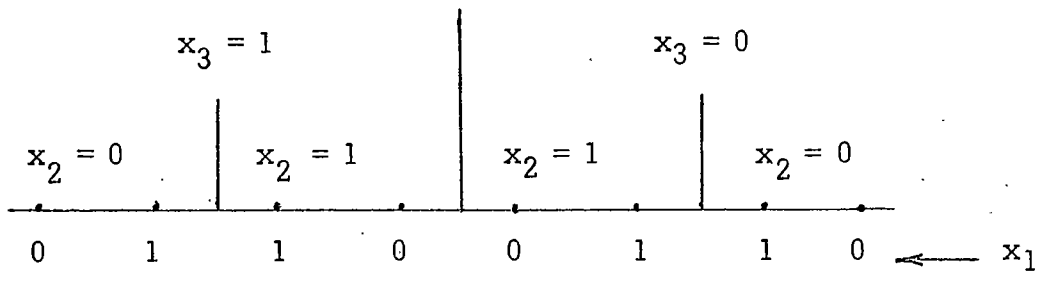


Figure 3.2.1 One Dimensional Graph Format for Three Variable Functions

$x_2 = 1$, are a mirror image of those in the right. This pattern is continued in the left-half portion having $x_3 = 1$. Each black dot on the horizontal line represents a different combination of values of the variables and therefore, stands for a minterm. A function may be plotted on this graph with upward vertical lines from the dots representing true minterms and downward vertical lines false minterms.

A function is called even symmetric if the true terms in one half of the one-dimensional graph occupy the same minterm positions with respect to the centre as the true terms on the other half. A function is odd symmetric if the true terms of one half of the graph occupy the same minterm positions with respect to the centre as the false terms on the other half. A function is one side flat if the true terms (false terms) are confined within one half of the graph.

A Maitra cascade is a one-dimensional array of two-input one-output cells where each cell can produce any logical function of two variables (Figure 3.2.2). If a function $f(x_n, x_{n-1}, \ldots x_1)$ can be realized by a Maitra cascade, then it is called Maitra-realizable (MR).

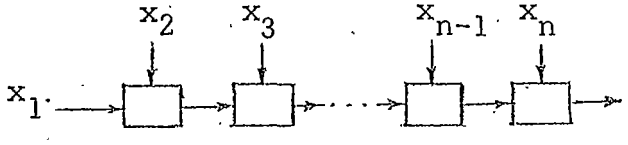


Figure 3.2.2 The Maitra Cascade

An MR function is realized with some ordering of inputs (in Fig. 3.2.2, the ordering is $x_n, x_{n-1}, \ldots x_1$) and appears with some characteristic pattern when plotted on a one-dimensional graph with this ordering of variables. The following property has been studied by Maitra but will be discussed in a modified form for subsequent reference.

Theorem 3.2.1 Let $f(x_n, \ldots x_1)$ be an MR function with the ordering: $x_n, x_{n-1}, \ldots x_1$. If it is plotted with this ordering, then f must be either i.) odd symmetric or ii) even symmetric or iii) one side flat.

Proof: The three cases mentioned in the theorem exhaust all possible situations for two variable functions. Any cell in the Maitra cascade has only two inputs. Considering the last cell of a cascade which realizes f, it is seen that f can be expressed as a function of two variables--x_n and some function $f_1(x_{n-1}, \ldots x_1)$. Hence the property must be true. Q.E.D.

Referring to Theorem 3.2.1, consider two possibilities. If case (i) or case (ii) is true, let $f_1(x_{n-1}, \ldots x_1)$ be the function in one half of the one-dimensional graph. If case (iii) is true let $f_1(x_{n-1}, \ldots x_1)$ be the function in the non-flat side (both sides flat is a trivial case). Since f is MR with ordering $x_n, x_{n-1}, \ldots x_1$, hence f_1 must be MR.

To see this, note that f can be expressed as $f = f_1 * x_n$ where $*$ is one of the sixteen two-variable functions. Therefore, either f_1 of $\bar{f}_1$ must be the output of the last but one cell in the Maitra cascade realizing f. It is known that complementary of an MR function is MR. Thus, f_1 must be MR. Therefore, f_1 must satisfy Theorem 3.2.1. In a similar manner, f_2 can be formed from f_1, f_3 from f_2---f_k from f_{k-1} for $k = 2,3,\ldots n-3$ and each must satisfy Theorem 3.2.1. This result gives a test procedure for MR function with an assumed ordering.

Theroem 3.2.2 There exists a Maitra-realizable function with k minterms true for every k such that $0 \leqq k \leqq 2^n$.

Proof: The proof is obtained by constructing an MR function of any given number of true minterms. A function with no minterms true is a trivial MR function. Consider the one-dimensional graph for n variables. One side flat MR functions of different size can be constructed by taking one minterm, two minterms, three minterms, etc. from one end as true terms. Q.E.D.

This theorem shows that the Maitra cascade has the necessary flexibility as stated in the proof of Theorem 2.2.6. Theorem 3.2.3 gives an upper bound on the number of necessary transpositions required by a Maitra cascade in order to realize an arbitrary function.

Theorem 3.2.3 To convert any given function into an MR function, at most $(2^{n-2} - 1)$ transpositions are necessary.

Proof: This property may be easily verified for 3-variable functions by using Maitra's one-dimensional graph and considering functions of up to 4 true minterms (because complementary of an MR function is also MR). According to the statement of the theorem, we need to use at most one transposition for conversion. Consider three cases for the 3-variable graph (Figure 3.2.1):

i) Two true minterms on each side of the origin.

ii) One true minterm on one side, Two true minterms on the other.

iii) No true minterm on one side.

Case (iii) is MR (one side flat). Case (ii) can be converted to MR function by interchanging the single true minterm on one side with a false minterm on the other. There may be three possibilities for case (i). The function may be odd symmetric or even symmetric and therefore MR. If none of these, there must be one pair of true terms which are mirror symmetric and one pair which are not. Clearly, a single transposition can make the function either odd or even symmetric and therefore MR. The other possible cases can be dealt with in a manner similar to one of these three cases.

Let us now consider n-variable functions. Assume that the theorem is true for (n-1)-variable functions. We shall then need consider only functions having m true minterms for $2^{n-2} < m \leqq 2^{n-1}$. For those functions having $m \leqq 2^{n-2}$, one of the halves of the graph will contain at most 2^{n-3} true terms. These can be interchanged with the false terms on the other half. When all of them are interchanged, at most 2^{n-3} transpositions will have been used. The resultant function will be one side flat and the other side--a function of (n-1) variables which need, according to the theorem, at most $(2^{n-3} -1)$ transpositions for conversion. In total then, $(2^{n-2} -1)$ transpositions can do the job.

Next we consider the special case when each half contains 2^{n-2} true terms. One of these will require at most $(2^{n-3} -1)$ transpositions for conversion into MR function. After one half has been converted into MR function, there will be on the other half at least 2^{n-3} true terms which will be a mirror image of the same number of true terms on the first half or at least the same number which will not be a mirror image. Therefore, at most 2^{n-3} transpositions will be necessary to get odd or even symmetry. Thus the total number of transpositions will be at the most $(2^{n-2} -1)$.

Next consider the case where one half (to be referred as A) contains $(2^{n-2} - K)$ true terms and the other half (B) contains $(2^{n-2} - L)$ true terms, where $K \geq L$ for K, L either zero or positive. The case of negative L will be discussed afterwards. Let p be the smallest integer such that $2^p \geq (K+L)$. The one-dimensional graph is divided into 2^{n-p} blocks of 2^p terms. If we mark these as 1,2,3,...from one end, there will be some s-th block (to be referred as S) of 2^p terms in B, which will contain at most

$$\left\lfloor \left(\frac{2^{n-2} - L}{2^{n-p-1}} \right) \right\rfloor = \left\lfloor \left(2^{p-1} - \frac{L}{2^{n-p-1}} \right) \right\rfloor \quad \text{true terms.}$$

The number of false terms in this block is at least

$$2^p - \left\lfloor \left(2^{p-1} - \frac{L}{2^{n-p-1}} \right) \right\rfloor \quad (\lfloor\ \rfloor \text{ denotes the base value})$$

If all of $(2^{n-2} - K)$ true terms of A are interchanged with the false terms in B, there will be $(K + L)$ false terms in B. Let this interchange be done in such a way that the 2^p terms of S are not disturbed before other false terms in B have been exhausted. After this has been done, the allowable number of transpositions remaining $= 2^{n-2} - 1 - (2^{n-2} - K) = K - 1$. The interchange may result in two possible situations.

Case I But for the s-th block, B is full of true terms.

In this case the function in S can be transformed into an MR function by at most (K-1) interchanges, as $K > 2^{p-2}$. By this, the over-all function will also be MR.

Case II There are some false terms in B outside S.

Let $2^{p-2} + f$ = Number of true terms in S, where $f < 2^{p-2}$.

(a) If $K \geq 2^{p-1}$, then all true terms of S may be interchanged with false terms — a sufficient number of these with all the false terms in B but outside S and the remaining true terms with the false terms within S in a manner such that a one-side flat MR function in S results. The over-all function is also MR in this case.

(b) Let $K = 2^{p-2} + c = L$ where $c < 2^{p-2}$. Values of K different from L will be considered later. Consider the following sub-cases.

(i) $c > f$. As in (a), all true terms in S can be interchanged with false terms and an MR function may be obtained.

(ii) $c \leq f$. Express c as $c = a_1 2^{p-3} + a_2 2^{p-4} + a_3 2^{p-5} + \ldots$, where a_i for $i = 1,2,3,\ldots$ is a binary valued variable, being either 0 or 1. Then $K = 2^{p-2} + a_1 2^{p-3} + a_2 2^{p-4} + \ldots$. Consider the terms in S: There will be one half of the s-th block containing less than 2^{p-2} true terms. With at most $2^{p-2} - 1$ interchanges, these are mapped (as many as needed) into the false terms in B outside S and the remaining in the rest of the s-th block. Next, choose the sub-block of 2^{p-2} terms in the other half of the s-th block containing a lesser number of true terms than the other sub-block. If each sub-block of 2^{p-2} terms in this half contains more than 2^{p-3} true terms, then since the total number of true terms in S is less than 2^{p-1}, hence in the preceding stage of the interchange, a correspondingly lesser number than $2^{p-2}-1$ interchanges were used.

Use this difference of interchanges to get a sub-block of 2^{p-2} terms, containing 2^{p-3} true terms or less. If $a_1 = 1$, then use at most 2^{p-3} interchanges to clear the sub-block of all true terms. The true terms can be mapped into the false terms in B outside S, or if there is none, into the false terms in S but outside the half already freed from true terms. If $a_1 = 0$, do not clear the sub-block of true terms. If $a_2 = 1$, choose that half of the sub-block of 2^{p-2} terms, which contains a lesser number of true terms and proceed in the manner stated before. Finally, an MR function with one side flat at each stage of its formation is obtained.

If K is greater than L, then the number of interchanges necessary to transfer true terms of A into B is reduced. Later in the process, the number of interchanges may be increased but the increase will be, at most, to the same extent as the reduction earlier.

The case of negative L may be considered now. If A and B contain 2^{n-2} -K and 2^{n-2} +L true terms respectively, with $K \geq L$, then, interchanging as before, and then bringing all false terms into S, the allowable number of interchanges remaining can be shown as

$$= K-1 - \left[(K-L) - \left\{2^p - \left\lfloor (2^{p-1} + \frac{L}{2^{n-p-1}}) \right\rfloor\right\}\right] = 2^p + L - 1 - \left\lfloor (2^{p-1} + \frac{L}{2^{n-p-1}}) \right\rfloor$$

This number is greater than $2^{p-2}-1$, needed to convert the function in S into MR. Q.E.D.

There exist functions which need at least $2^{n-2} - 1$ transpositions to be MR.

An example is $F = S_2(x_1, x_2, x_3, x_4)$, where S_2 indicates that F is true if and only if two variables out of four are true. It shows that the bound on the required number of transpositions is the least upper bound. It also shows that resorting to different ordering of variables cannot improve the bound, because the function is symmetric.

3.3 Determination of Necessary Transpositions

The proof of Theorem 3.2.3 gives a method of finding the necessary maps to realize an arbitrary function with a bulk transfer system having its output logic made up of a flexible Maitra cascade. Given a function, plot it on a one-dimensional graph. Next, determine the transpositions necessary to convert the function into MR function in the manner described under Theorem 3.2.3. Now set the mapping device to produce the composite map made up of the determined transpositions and set the output logic to produce the MR function into which the given function has been transformed.

An example of converting a non-Maitra-realizable function into MR function is worked out, giving the different steps.

Example 3.3 $F = S_2(x_4, x_3, x_2, x_1)$

Step 1. Plot the function on a graph (Figure 3.3.1)

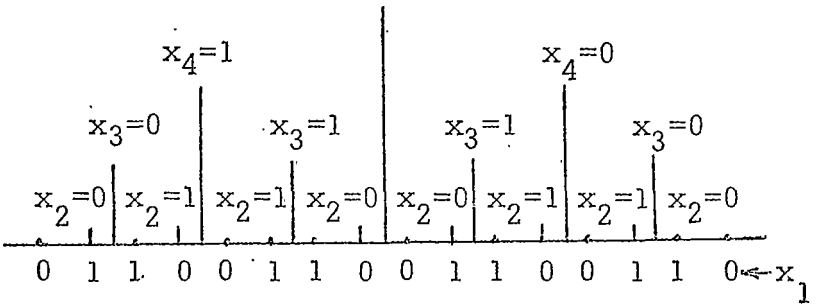


Figure 3.3.1 Plot of $F = S_2(x_4, x_3, x_2, x_1)$ in One-dimensional Graph.

Only true terms are shown by upward vertical line. There are 3 such terms on each half.

Step 2. Determine which of the cases in the Theorem 3.2.3 apply, i.e., the special case of 2^{n-2} true terms on each side or the more general case.

General case; K = 1, L = 1. True terms of any side can be mapped into the false terms of the other. Choose the right half true terms for this.

Step 3. Determine p from $2^p \geq K + L$.

We have $2^p \geq 2$. Thus the lowest value of p is 1.

Step 4. Mark the blocks of 2 terms starting from the left-most term to find a block containing minimum number of true terms.

Third block from the left. It contains two false terms.

Step 5. Interchange true terms of the right-half with the false terms of the left-half, keeping the false terms of the third block unchanged as long as there are other false terms in the left-half.

Except for third block, all other terms are now true in left-half.

Step 6. Interchange the false terms of the left-half excluding the third block with true terms of the third block.

Step 7. Convert the function in the third block into MR function by the remaining interchanges.

Steps 6 and 7 do not make any modifications. The resultant function is shown in Figure 3.3.2.

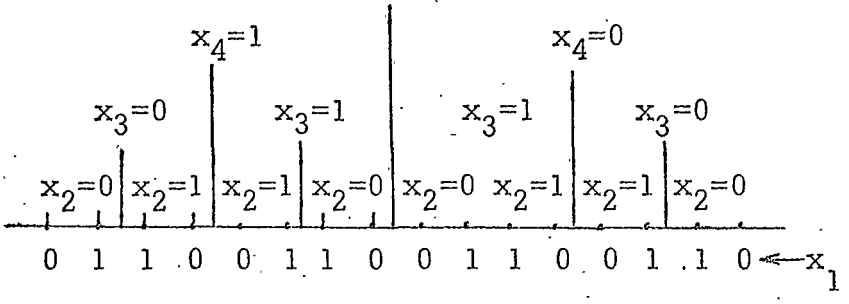


Figure 3.3.2 The Function $F = S_2(x_4, x_3, x_2, x_1)$ after Application of Suitable Transpositions.

The required number of transpositions obtained in this example is the minimum number possible. But this is not necessarily true in all cases. Sometimes a simple inspection may reveal fewer transpositions to make a function Maitra-realizable. The required number of transpositions in a particular case may be applied from the input array to the output array with the help of a flexible mapping device capable of implementing any transposition in accordance with the scheme described in section 2.1 of Chapter 2. An alternative is to place an array of mapping elements between input and output array, each element being capable of implementing any transposition. Such a flexible mapping element is shown in Figure 3.3.3. The pair of cascades with flexible logic cells, shown in the center of the Figure, are used to detect the occurrence of the pair of minterms between which an interchange is necessary. This is done by setting the cell functions to produce complementation in proper variables and gating the outputs of the cells through an AND block which produces a true output only if the desired combination of inputs to the cascade is present. The output of the OR gate is true when anyone of the pair of minterms is present; it is false otherwise. A true output from the OR gate is used to complement proper variables in the topmost cascade, with the cells there having been set earlier to produce suitable functions of their inputs.

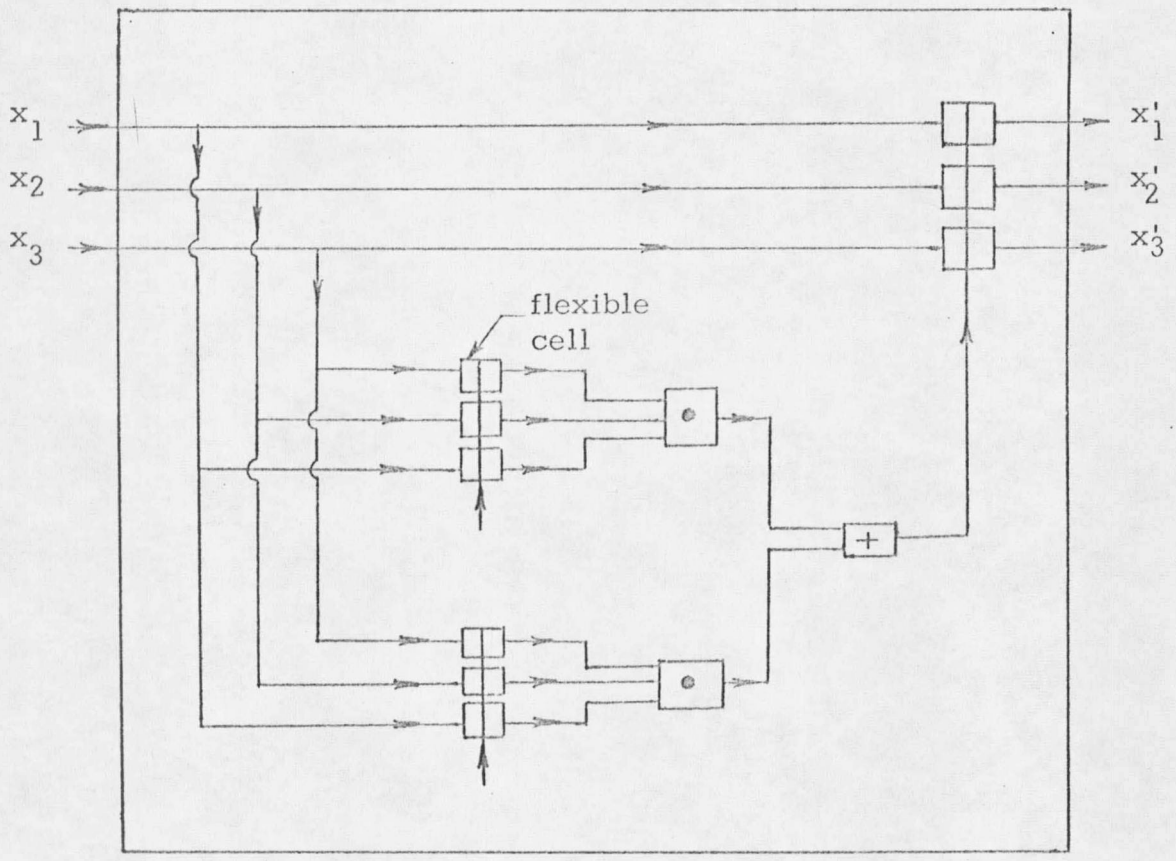


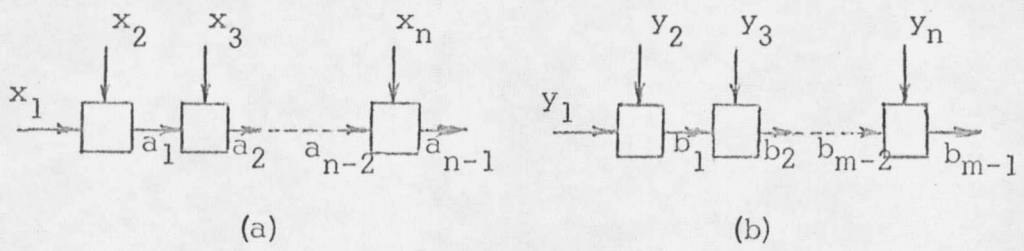
Figure 3.3.3 Flexible Mapping Element for Transposition Maps (3-variable)

With the type of mapping element just discussed, it is simple to produce many-to-one maps of the set of minterms. First, define a function with the group of minterms, all of which are to be mapped onto one minterm. Next, realize the function employing any suitable network and use the output from this network in the same manner as the output of the OR gate in Figure 3.3.3. Many-to-one mapping of

the set of minterms can be less complex than permutation of the set and may require lesser number of mapping operations in suitable cases. Example 3.3 is an instance of such case. Using only permutation maps, it was shown that three interchanges were necessary to convert the given function into an MR function. When many-to-one maps are allowed, it is possible to augment the given function by including the minterms --- $x_1x_2x_3x_4$ and $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$ --- so that an MR function is obtained. The output logic is set to realize this augmented function. To produce the desired logic, however, mapping device can be set earlier to map these two terms to some false term, such as $\bar{x}_1\bar{x}_2\bar{x}_3x_4$ of the function realized by the output logic.

3.4 Bulk Transfer in Cascades

Consider the Maitra cascade shown in Figure 3.4.1.a. One may regard the set of direct inputs $\{x_1, x_2, \ldots x_n\}$ along with the set of cell outputs $\{a_1, a_2, \ldots a_{n-1}\}$ as a new set of variables which can be treated in the same manner as the variables of the input array of a bulk transfer system discussed earlier. Similarly, the output array may be another array consisting of the direct inputs $\{y_1, y_2, \ldots y_m\}$ to a number of cells either in the same cascade or in a different cascade along with the outputs $\{b_1, b_2, \ldots b_{m-1}\}$ of those cells. The mapping device may consist of a mechanism for



(a) (b)

Figure 3.4.1 (a) Input Cascade (b) Output Cascade

repositioning of the variables so that only logic-free maps as discussed previously can be implemented. The set $\{y_1, y_2, \ldots y_m\}$ mapped into the output array is a subset of the set of variables $\{x_1, \ldots x_n, a_1, \ldots a_{n-1}\}$ and is transformed in the cascade to produce a new set of dependent variables $\{b_1, \ldots, b_{m-1}\}$. The mapping back from the output array into the input array need not be identity mapping. For mapping from one array into another, a proper subset of the set of independent and dependent variables may take part. A proper subset is necessary because the number of direct inputs in a cascade is always less than the sum total of the independent and dependent variables. The mapping is assumed to be one-to-one in the sense that each member of the subset is mapped to a different (direct) cell input. Any one of the cascades may serve as the output logic device. The cascades may be logically flexible and can be reset after each mapping or may be logically fixed. The final output function is obtained as one member of an array at the end of a suitable number of mappings. It is clearly possible to have several output functions appear

in the different members of the final array. Further, one may have more than one cascade in one array.

Let us consider a simple and specific case. Let the input array consist of the variables $\{ x_1, x_2, \ldots x_n, a_1, \ldots a_{n-1} \}$ and the output array of the variables $\{ y_1, \ldots y_m, b_1, \ldots b_{m-1} \}$. Let the mapping be logic-free and the output logic consist of the input and output cascades. Let these cascades be logically flexible.

One may show that such a system has logical universality if the input and output cascades contain at least one extra direct cell input in addition to the total number of original inputs $\{ x_1, x_2, \ldots x_n \}$. To see this, consider any function of n variables. A term of the function may contain, at most n literals and can be realized in a cascade of (n-1) cells, each realizing either an AND(OR) or a NIMP(IMP) function. Let the term realized be represented by a_n. After realizing the term in the input cascade, a mapping may be performed such that the variables $\{ x_1, x_2, \ldots x_n \}$ are mapped identically into the output cascade array while the variable a_n is mapped to the extra cell input y_{n+1} placed at the end of the output cascade. By setting the proper cell functions in the output cascade, a second term of the function may be realized and combined with the

first term by having the last cell in that cascade perform an OR(AND) function. Mapping back now, one may realize a third term and in this manner the procedure may be continued until all the terms have been realized.

This type of B.T. system has an equivalent in the Two-rail Cascade of Short[16]. While an extra arterial connection is used in the Two-rail Cascade to propagate the partially-formed function, an extra cell input is used here to represent the partially-formed function and combine it to the newly-formed term. As possible in the Two-rail Cascade also, one may realize a suitable collection of terms in one cascade at a time, so that the number of times mapping has to be used can be reduced to an extent depending on how the given function may be decomposed into cascade-realizable functions.

No use has been made in the above argument of the intermediate cell functions a_i(or b_i) for $i < n$. It is possible to have extra cell inputs where necessary to represent and use them in forming a part or the whole of the function in a more efficient manner. In the following, an example of realizing a given function using bulk transfer in cascades is worked out.

Example 3.4 $F = x_1(x_2 + x_3) + (x_2 + x_3 + x_4)\, x_5 + x_2\, x_3\, x_6$

This function may be realized in accordance with the procedure suggested earlier by realizing one term of the function in a cascade, mapping the output to the other cascade and combining with the next term realized in that cascade, mapping back and so on. Figure 3.4.2

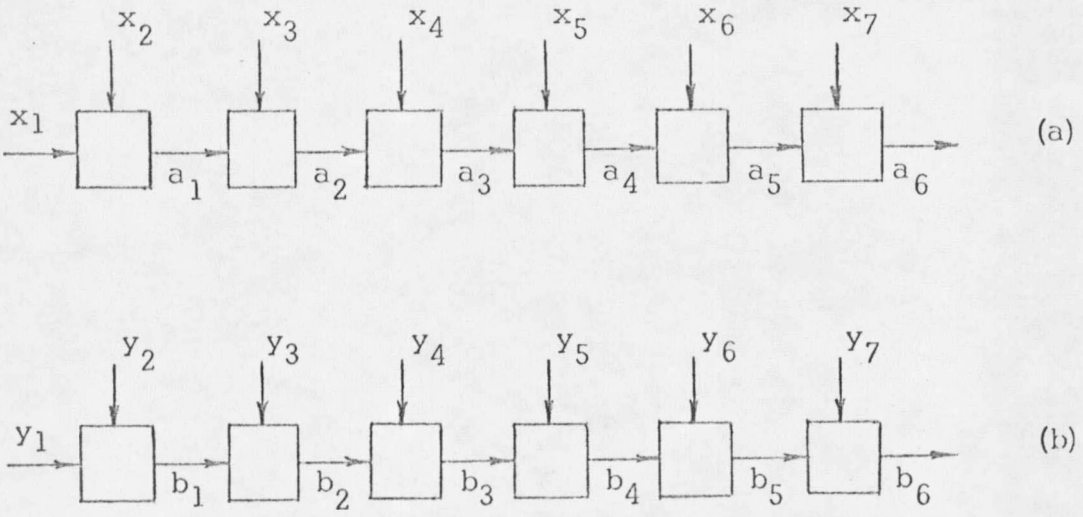


Figure 3.4.2 (a) Input Cascade (b) Output Cascade

shows the two cascades. The cell inputs $\{y_1, y_2, \ldots y_6\}$ are the same as the inputs $\{x_1, x_2, \ldots x_6\}$. The cell input $y_7 = a_6$ and $x_7 = b_6$. The cell functions in the input cascade are initially

$$a_1 = x_1 x_2$$

$$a_6 = a_5 = a_4 = a_3 = a_2 = a_1$$

To realize the second term, map the input array into output array as specified. The cell functions in the output cascade are initially

$$b_1 = y_1 = x_1$$

$$b_2 = b_1 y_3 = x_1 x_3$$

$$b_5 = b_4 = b_3 = b_2$$

$$b_6 = b_5 + y_6 = x_1 x_3 + x_1 x_2$$

To realize the next term, change appropriately the cell functions in the input cascade and map the output array back as specified above. The cell functions in the input cascade are now

$$a_1 = x_2$$

$$a_3 = a_2 = a_1$$

$$a_4 = x_5 a_3 = x_5 x_2$$

$$a_5 = a_4$$

$$a_6 = x_7 + a_5 = x_1 x_3 + x_1 x_2 + x_5 x_2$$

In this manner, to realize the function, five mappings are necessary. To illustrate that suitable use of intermediate functions can reduce the number of mappings to a considerable extent, another realization will be given below. The input and output cascades are shown in Figure 3.4.3.a and 3.4.3.b.

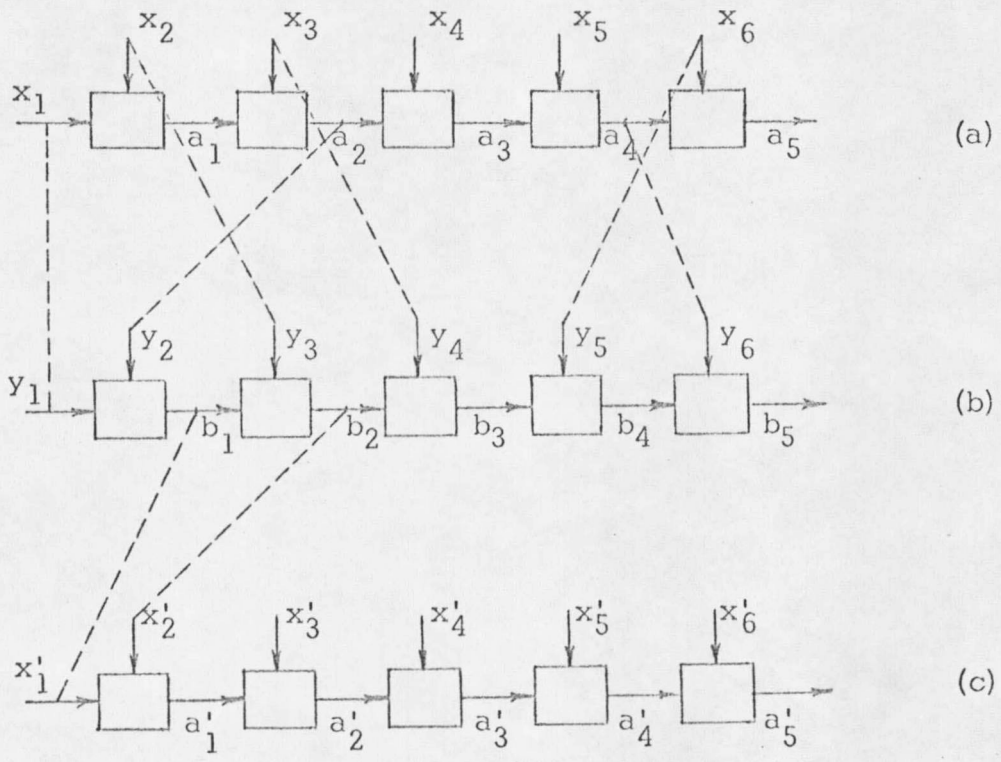


Figure 3.4.3 Array Configuration on Mapping for the Realization of $F = x_1(x_2 + x_3) + (x_2 + x_3 + x_4)\, x_5 + x_2x_3x_6$.
(a) Input Cascade Array (b) Output Cascade Array (after a mapping) (c) Input Cascade Array (after two mappings).

Let the cell functions in the input cascade be set to produce the following functions (Figure 3.4.3.a):

$$a_1 = x_2$$

$$a_2 = x_2 + x_3$$

$$a_3 = x_2 + x_3 + x_4$$

$$a_4 = x_5 (x_2 + x_3 + x_4)$$

$$a_5 = x_6$$

Next, use a suitable map to have (Figure 3.4.3.b)

$$y_1 = x_1$$

$$y_2 = a_2$$

$$y_3 = x_2$$

$$y_4 = x_3$$

$$y_5 = x_6$$

$$y_6 = a_4$$

Then, set the cell functions to have (Figure 3.4.3.b)

$$b_1 = y_1 y_2 = x_1 (x_2 + x_3)$$

$$b_2 = y_3 = x_2$$

$$b_3 = y_3 y_4 = x_2 x_3$$

$$b_4 = y_3 y_4 y_5 = x_2 x_3 x_6$$

$$b_5 = b_4 + y_6 = x_2 x_3 x_6 + x_5 (x_2 + x_3 + x_4)$$

Let x_i' be the variable mapped back for the first time to the previous input position x_i. Use suitable map (Figure 3.4.3.c) so that

$$x_1' = b_1 = x_1 (x_2 + x_3)$$

$$x_2' = b_5 = x_2x_3x_6 + x_5 (x_2 + x_3 + x_4)$$

Other variables are no longer necessary. Now, set the cell function such that

$$a_1' = x_1' + x_2' = x_1 (x_2 + x_3) + x_2x_3x_6 + x_5 (x_2 + x_3 + x_4)$$

$$= F$$

It is possible to realize the above function in several other ways. An efficient realization is largely dependent on a suitable decomposition of the function.

Chapter 4

PARALLEL BULK TRANSFER SYSTEM

4.1 Parallel Transfers

It has been shown in Chapter 2 that arbitrary logic can be realized using bulk transfer technique. To do this, complex maps of the input array are sometimes required and composed out of a limited number of basic maps available in the mapping device. As it is a lengthy and time-consuming process, it is necessary to consider a serio-parallel approach to the problem. One may divide the data on the input array into smaller sub-arrays and then, on these sub-arrays bulk transfer technique can be applied in parallel. The general nature of the system in this scheme can be visualized as a multi-level B.T. system with a tree-like structure (Figure 4.1).

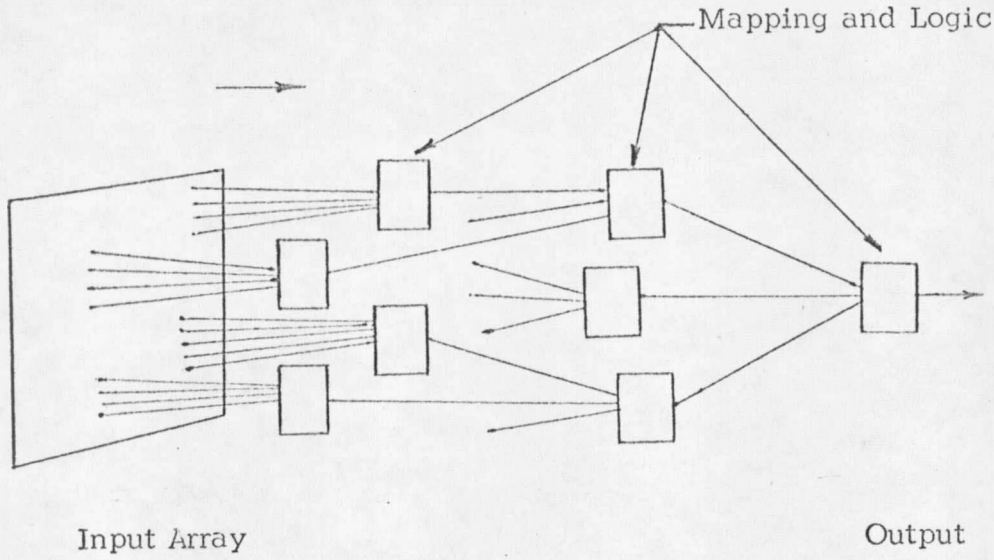


Figure 4.1 Multi-level Bulk Transfer System

To investigate the characteristics of this type of parallel B.T. system, we face the following problems: Suppose we are given an input array of variables. We partition the array into a set of disjoint or partially joint sub-arrays and realize arbitrary functions on the variables of each of the sub-arrays to get a smaller array of outputs which we subject to the same treatment as before and then repeat the process until a single function is realized. What kind of functions can be produced in this manner? Given a function, how do we set up the system to realize it? If all functions are not realizable, how do we test for the realizability of a function? These questions generally pertain to the subject of decomposition of switching functions.

The subject of decomposition of switching functions has been studied in detail by Ashenhurst[2] and Curtis[4]. However, the type of decomposition to be considered here, though restricted, presents a formidable problem because of sheer size when these methods are sought to be applied. These are chart methods for the purpose of general decomposition. The problem associated with decomposition of a large array into joint or disjoint sub-arrays of known size and geometry and study of the realizable functions seems impossible to handle in terms of decomposition charts and needs some

sort of algebraic approach. Roth and Karp[15] have used this kind of approach in dealing with the problem of minimization of Boolean graphs. The following matter will be developed along the latter line.

A bulk transfer system may be compared with respect to logical universality with the recently proposed Universal Logical Module[5] or ULM[1]. The term "ULM" will be used in the following for a bulk transfer system or any other device having logical universality, because the theoretical development is independent of how the device is implemented. Consider the input array of the B.T. system in Figure 2.1.1. Let us assume that the array of cells is divided into s disjoint subsets where each subset does not necessarily contain the same number of cells. Also assume that there are a set of ULM's each of which can realize arbitrary logical functions on the variables of each subset and, further, the outputs of these ULM's are similarly (as in the original array) fed as input to next higher level ULM's and the process is repeated until a single function is realized. (Note that the inputs to an ULM at any level come from a disjoint set of outputs of the previous level and these sets need not be of the same size). Clearly if the logical functions performed by each ULM is known, the ultimate

---

[1]According to conventional terminology, an n-variable ULM has, in general, m inputs where $m \geq n$. Since we are not concerned with the structure of the ULM, we shall mean, by n-input ULM, a ULM of n variables.

function can be determined in terms of the original variables by tracing back from the output to the input. But the reverse problem--given a function, to decide if it is realizable by such a scheme, and when realizable, to find the functions to be performed by each ULM--is not so simple. To know the realizable functions we must find a criterion for realizability. Realizability tests for a multilevel system, where each level contains a number of multi-input ULM's will therefore be developed here. The algorithm to be presented has some advantages because of the fact that it is applicable to the sum-of-prime-implicants form of a function so that the vast number of minterms of the function need not be considered. The sum-of-prime-implicants form is assumed to be obtained from an original disjunctive-form expression by some standard procedures such as the method of consensus[10]. Further, the algorithm combines the advantages of algebraic method with those of tabular technique. Finally, the complexity of the algorithm does not increase with the size of subsets into which the array is originally partitioned. However, computations increase with the number of inputs of a ULM.

The case where the partitioning of the array is such that the resultant sub-arrays are not disjoint will be considered in a later section. As may be guessed, this case is more complex, particularly

because of the fact that there are more than one way of realizing certain functions at any level and not all of these may ultimately lead to a realization.

4.2 Disjoint Decomposition

In this section we shall start by considering simple cases of disjoint decomposition and gradually build up more complex case. A general algorithm for testing realizability will be given at the end.

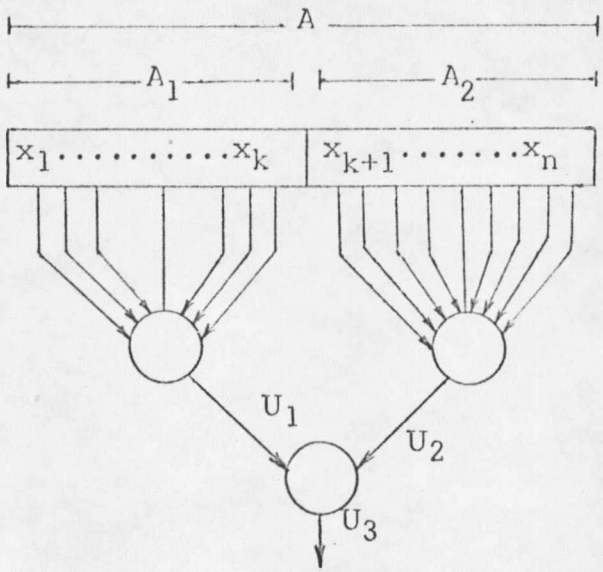


Figure 4.2.1 Two-level Network with Two-input ULM at the Last Level

Consider the following simple case: An array A is partitioned into disjoint subsets A_1 and A_2 (Figure 4.2.1). U_1 is any arbitrary function on A_1 and U_2 on A_2 and U_3 is the final output function. Let F be a given function expressed as a sum of prime implicants. If F can be realized by the network of Figure 4.2.1, then F can be expressed as:

$$F=U_3(U_1, U_2)=U_3\{ U_1(x_1,x_2,\ldots,x_k), U_2(x_{k+1},\ldots,x_n)\}$$

As the decomposition is disjunctive, the occurrence of the variables $x_1, x_2,\ldots,x_k$ in F must be accounted for by the function U_1 and the occurrence of the variables $x_{k+1}, x_{k+2},\ldots,x_n$ by the function U_2. Therefore we can make a decomposition table (Table 4.2.1) with two columns headed by A_1 and A_2.

Table 4.2.1 Decomposition Table

A_1	A_2
P_1	P_2
Q_1	Q_2
.	.
.	.
.	.

Each prime implicant (PI) of F occupies a distinct row in this table. Those literals of the PI that are formed of the variables of A_1 are placed

under A_1 and those formed of the variables of A_2 are placed under A_2, both parts occupying the same row. For notation, a subscript 1 will be used for a product term under A_1 such as P_1, Q_1 etc. Similarly, a subscript 2 will be used for a product term under A_2, such as P_2, Q_2 etc. In general, therefore, PI's of F will be represented as P_1P_2, P_1Q_2, Q_1Q_2 etc. where P_1, P_1, Q_1 appear under A_1 and P_2, Q_2, Q_2 appear under A_2.

We shall assume U_3 to be a non-degenerate function of U_1 and U_2, as otherwise, F must be independent of the variables of one or both of these functions. We shall consider here a linear array of sub-arrays, but there is no loss of generality because, whatever be the dimension of the original array, since the decomposition is disjunctive, the sub-arrays can always be arranged in a linear form. First, a few theorems will be proved on which the algorithm will be based. The theorems are true for any realizable function F.

Theorem 4.2.1 Let P_1P_2 be a prime implicant of F. Then P_1 is a PI of either U_1 or $\overline{U}_1$ and P_2 is a PI of either U_2 or $\overline{U}_2$.

Proof: Let P_1P_2 be a PI of F. We shall first prove that P_1 is either in U_1 or in $\overline{U}_1$ and P_2 is either U_2 or in $\overline{U}_2$.

The following are the possible inclusion situations for P_1 and P_2.

A. For P_1:

i) $P_1 \subseteq U_1$

ii) $P_1 \subseteq \overline{U}_1$

iii) $P_1 \not\subseteq U_1$, $P_1 \not\subseteq \overline{U}_1$ and there exist minterms M_1 and M_2 contained in P_1 such that $M_1 \subseteq U_1$, $M_2 \subseteq \overline{U}_1$.

B. For P_2:

1) $P_2 \subseteq U_2$

2) $P_2 \subseteq \overline{U}_2$

3) $P_2 \not\subseteq U_2$, $P_2 \not\subseteq \overline{U}_2$ and there exist minterms M_3 and M_4 contained in P_2 such that $M_3 \subseteq U_2$, $M_4 \subseteq \overline{U}_2$.

Contrary to our assertion, assume the following is true:

Case I iii) is true for P_1 and 3) is true for P_2. Assign values to the inputs $(x_1, \ldots, x_n)$ such that the following truth table is obtained.

Input Assignment	U_1	U_2	F
$M_1 = 1$, $M_3 = 1$	1	1	1
$M_1 = 1$, $M_4 = 1$	1	0	1
$M_2 = 1$, $M_3 = 1$	0	1	1
$M_2 = 1$, $M_4 = 1$	0	0	1

F is seen to be a constant function which contradicts the hypothesis that F is non-degenerate.

Case II i) is true for P_1 and 3) is true for P_2.

Assign values to the inputs such that $P_1 = 1$ and hence $U_1 = 1$. Since $P_2 = 1$ whenever $M_3 = 1$ or $M_4 = 1$, this means that F is true whenever $U_1 = 1$. Hence $F = U_1 + S(U_1, U_2)$ where $S(U_1, U_2)$ is some function of U_1 and U_2. But $P_1 \subseteq U_1$, therefore

$$U_1 = P_1 + g(x_1, \ldots, x_k) \qquad \text{for some function } g(x_1, \ldots, x_k)$$

$$F = P_1 + g(x_1, \ldots, x_k) + S(U_1, U_2)$$

This implies that P_1 is an implicant of F. This contradicts the assumption that P_1P_2 is a PI of F.

Case III ii) is true for P_1 and 3) is true for P_2.

Arguing as in case II, we can show that $F = P_1 + h(x_1, \ldots, x_k) + R(U_1, U_2)$ which gives a contradiction.

Case IV and V, as stated below, are analogous, since we shall arrive at the contradiction that P_2 is an implicant of F.

Case IV iii) is true for P_1 and 1) is true for P_2.

Case V iii) is true for P_1 and 2) is true for P_2.

The remaining four cases conform to the theorem.

Case VI $P_1 \subseteq U_1$ and $P_2 \subseteq U_2$.

Case VII $P_1 \subseteq \overline{U}_1$ and $P_2 \subseteq U_2$.

Case VIII $P_1 \subseteq U_1$ and $P_2 \subseteq \overline{U}_2$.

Case IX $P_1 \subseteq \overline{U}_1$ and $P_2 \subseteq \overline{U}_2$.

The theorem will be proved if we can show that in these last four cases P_1 is a PI of U_1 or $\overline{U_1}$ and P_2 is a PI of U_2 or $\overline{U_2}$. In case VI,

$$F = U_1U_2 + G(U_1, U_2)$$

since $P_1 P_2$ is a PI of F, and $G(U_1, U_2)$ is some function in U_1 and U_2. Suppose P_1 is not a PI of U_1. Thus there exists a PI of U_1, say P_1^* such that $P_1 \subset P_1^*$. But when $P_1^* = 1$, $P_2 = 1$ we have $U_1U_2 = 1$, that is $F = 1$. Thus P_1P_2 cannot be a PI of F. A contradiction. Hence P_1 must be a PI of U_1. In a similar way, we can show that P_2 is a PI of U_2. In an analogous fashion we can prove that in case VII, P_1 is a PI of $\overline{U_1}$, P_2 is a PI of U_2, in case VIII, P_1 is a PI of U_1 and P_2 is a PI of $\overline{U_2}$ and in case IX, P_1 is a PI of $\overline{U_1}$ and P_2 is a PI of $\overline{U_2}$. Hence the theorem. Q.E.D.

Theorem 4.2.2 If P_1P_2 and Q_1Q_2 are two PI's of F and if $P_2 = Q_2$ then $Q_1 \subseteq U_1(\overline{U_1})$ if $P_1 \subseteq U_1(\overline{U_1})$ and further, P_1 and Q_1 are PI's of $U_1(\overline{U_1})$. Conversely, if $P_1, Q_1 \subseteq U_1(\overline{U_1})$, then $Q_1P_2 \subseteq F$ if $P_1P_2 \subseteq F$.

Further, if P_1 and Q_1 are PI's of $U_1(\overline{U_1})$ and P_1P_2 is a PI of F, then Q_1P_2 is a PI of F.

Proof: First Part: Let P_1P_2 and Q_1Q_2 be two PI's of F such that $P_2=Q_2$. Further, WLOG, let $P_1 \subseteq U_1$ and $P_2 \subseteq U_2$. If possible, let $Q_1 \subseteq \overline{U_1}$. Consider the input conditions such that P_1P_2 is true. Then,

$F = U_3(U_1, U_2) = 1$.

Expressed in terms of U_1, U_2, and U_3,

when $U_1 = 1$, $U_2 = 1$, then $U_3 = 1$.....(i)

Change the input conditions as necessary to make Q_1 true. Since $P_2 = Q_2$, we have now Q_1Q_2 true and consequently $F = 1$. Expressing in terms of U_1, U_2, and U_3,

when $U_1 = 0$, $U_2 = 1$, then $U_3 = 1$...(ii)

From (i) and (ii), we can write $F = U_2 + T$ where T is some expression involving, in general, U_1 and U_2. Also, since $P_2 \subseteq U_2$, $U_2 = P_2 + S$ where S is some sum of product terms of the original variables. Therefore,

$$F = P_2 + S + T.$$

It follows that P_1P_2 is not a PI of F. Contradiction.

Therefore if $P_1 \subseteq U_1$, then $Q_1 \subseteq U_1$. Similarly it can be shown that if $P_1 \subseteq \overline{U_1}$ then $Q_1 \subseteq \overline{U_1}$. That P_1 and Q_1 are PI's of $U_1(\overline{U_1})$ follows

from Theorem 4.2.1.

Proof of the converse: Let P_1, $Q_1 \subseteq U_1$ and $P_1P_2 \subseteq F$. Assume WLOG, $P_2 \subseteq U_2$. Consider the input conditions such that P_1P_2 is true. Therefore,

$$F = U_3(U_1, U_2) = 1.$$

Expressing in terms of U_1, U_2 and U_3, therefore, we may write,

$$F = U_1U_2 + T$$

where T is again some expression involving U_1 and U_2.

Now change the input conditions as necessary to make Q_1 true. The true condition of P_2 need not be changed for this purpose. As $U_1 = 1$, $U_2 = 1$, hence from above, $F = 1$. Therefore $Q_1P_2 \subseteq F$ if $P_1P_2 \subseteq F$.

Let P_1 and Q_1 be PI's of U_1 and P_1P_2 a PI of F. Therefore, $Q_1P_2 \subseteq F$. If Q_1P_2 is not a PI, let $Q_1{*}P_2{*}$ be a PI of F where

$$Q_1 \subseteq Q_1{*}$$

and $$P_2 \subseteq P_2{*}$$

such that at least one of the implication relations is proper.

Let $Q \subset Q_1{*}$. We then have $Q_1P_2 \subset Q_1{*}P_2 \subseteq F$. Then, using first part, $Q_1{*} \subseteq U_1$. Therefore Q_1 cannot be a PI of U_1. Contradiction.

So $Q_1 = Q_1^*$. Let $P_2 \subset P_2^*$. Then,

$$P_1P_2 \subset P_1P_2^* \subseteq F \text{ (since } Q_1P_2^* \subseteq F).$$

So P_1P_2 cannot be a PI of F. Contradiction.

Hence $P_2 = P_2^*$. That is, with the given conditions, Q_1P_2 is a PI of F.

Q.E.D.

Corollary 1. The set of PI's of U_1 and $\overline{U_1}$ obtained under A_1 completely define U_1 and $\overline{U_1}$. Similarly the set of PI's obtained under A_2 completely define U_2 and $\overline{U_2}$.

Proof: If the original expression for F is a complete sum of PI's then the corollary follows from the converse of Theorem 4.2.2. When U_3 is not unate, in say, U_1 or $\overline{U}_1$, then all PI's of U_1 and $\overline{U}_1$ will appear under A_1. If U_3 is unate, in, say U_1, then the PI's of $\overline{U}_1$ can be obtained from U_1 by complementation.

Let F be not a complete sum of PI's. Let $M_1 \subseteq U_1$ be a minterm that is not included in the PI's of U_1 occurring under A_1. Let P_1P_2 be a PI of F such that $P_1 \subseteq U_1$. Then, by Theorem 4.2.2, $M_1P_2 \subseteq F$. Hence, M_1P_2 must be included in some PI's of F. By assumption P_2 is not a PI of F. Hence $\exists Q_1$ under A_1 such that $M_1P_2 \subseteq Q_1P_2$. It follows that $M_1 \subseteq Q_1$. By Theorem 4.2.1, Q_1

is a prime implicant of U_1. Thus, the assumption $M_1 \subseteq U_1$, but not included in a PI of U_1 appearing under A_1 is false. Similarly, it can be shown that $\overline{U_1}$ is completely defined by the terms under A_1 and U_2 and $\overline{U_2}$ are completely defined by the terms under A_2.

Q.E.D.

Theorem 4.2.3 Let $U_1(x_1,\ldots,x_k)$ and $U_2(x_{k+1},\ldots,x_n)$ be two Boolean functions defined on disjoint subsets of variables. If U_1 and U_2 are expressed as a sum of prime implicants like $U_1 = P_1 + Q_1 +\ldots$, and $U_2 = P_2 + Q_2 +\ldots$, then each resultant product term when

$$F(x_1,\ldots,x_k) = U_1 U_2 = (P_1 + Q_1 +\ldots)(P_2 + Q_2 +\ldots)$$

is written as a sum of products is a prime implicant of F.

Proof: With the conditions given in the above theorem, let P_1P_2 and Q_1Q_2 be two distinct product terms of F. From the definition of prime implicant it follows that among the product terms of F, if a relationship is found between a pair of terms such that (i) one term is of the form Rx_i and the other term of the form $S\overline{x_i}$ where R and S are products of literals not involving x_i and either $R \subseteq S$ or $S \subseteq R$ or, (ii) one term is subsumed by the other, then and then only, all terms of F do not represent prime implicants.

As P_1P_2 and Q_1Q_2 can be any two general terms of F, we need to show that case (i) and (ii) cannot occur between P_1P_2 and Q_1Q_2.

Let us suppose, case (i) is true. WLOG, let $x_i \subseteq \{x_1, \ldots x_k\}$ and $P_1P_2 = x_iP_1{}^*P_2$ and $Q_1Q_2 = \bar{x}_iQ_1{}^*Q_2$ with $P_1 = x_iP_1{}^*$ and $Q_1 = \bar{x}_iQ_1^*$ where $P_1{}^*$ and $Q_1{}^*$ are the products of literals remaining after deletion of x_i and $\bar{x}_i$ from P_1 and Q_1 respectively. Also assume, WLOG, that $P_1{}^*P_2 \subseteq Q_1{}^*Q_2$. Now, $\{P_1, Q_1\}$ and $\{P_2, Q_2\}$ are defined on disjoint sets of variables. Therefore, we must have,

$$P_1{}^* \subseteq Q_1{}^*$$

and $$P_2 \subseteq Q_2$$

It follows that U_1 is not a sum of prime implicants because case (i) is satisfied between P_1 and Q_1. This is contradictory to the assumed conditions. Therefore case (i) cannot be true about P_1P_2 and Q_1Q_2.

To prove that case (ii) cannot possibly occur, it is sufficient to show that there exists at least one minterm in P_1P_2 which does not occur in Q_1Q_2 and vice versa.

Since P_1P_2 and Q_1Q_2 are distinct, either $P_1 \neq Q_1$ or $P_2 \neq Q_2$ or both. WLOG, assume $P_1 \neq Q_1$. As P_1 and Q_1 are PI's of U_1, one is not subsumed by the other. Therefore

$$\exists \; P_1 X_i \not\leq Q_1 \text{ and}$$

$$Q_1 X_j \not\leq P_1$$

where X_i, X_j are products of the literals of some variables in $\{x_1, \ldots, x_k\}$. It follows therefore that

$$P_1 X_i P_2 \not\leq Q_1 Q_2 \text{ and}$$

$$Q_1 X_j Q_2 \not\leq P_1 P_2.$$

Therefore case (ii) also cannot occur. Hence the theorem,

Q.E.D.

The following theorem can be proved very simply for the two-variable case. However, for ease in generalization, it is proved in somewhat detail.

Theorem 4.2.4 U_3 is unate in $U_1(U_2)$ or $\overline{U_1}(\overline{U_2})$ if and only if the sum of the terms under $A_1(A_2)$ is not equal to 1.

Proof: First Part: WLOG, let U_3 be unate in U_1 and let $\sum_i P_1^i$ = sum of the terms under $A_1 = 1$, where P_1^i represents some general term under A_1. We shall show a contradiction in this assumption. We have

$$U_1 = f(x_1, \ldots, x_k) \neq 1 \text{ or } 0 \text{ (as F is non-degenerate) and}$$

$$\sum_i P_1^i = 1 = U_1 + \overline{U}_1.$$

The first condition implies the presence of some product of the literals of the variables of A_1 in at least one PI of F. The second condition implies that there must be at least a pair of these products so that the sum may be 1 and further that the products must belong to complementary functions as $U_1 \neq 1$ or 0. It follows that there exists at least a pair of PI's of F, P_1P_2 and Q_1Q_2 such that $P_1 \subseteq U_1$ and $Q_1 \subseteq \overline{U_1}$. WLOG, let $P_2 \subseteq U_2$. At first, consider $Q_2 \subseteq U_2$. When P_1P_2 is true, then $U_1 = 1$, $U_2 = 1$ and $U_3 = 1$. When Q_1Q_2 is true, then $U_1 = 0$, $U_2 = 1$, and $U_3 = 1$. Therefore,

$$U_3 = U_2 + T$$

where T is some expression involving U_1 and U_2 in general. But

$$U_2 = Q_2 + P_2 + S \qquad \text{(from earlier assumption)}$$

when S is some expression involving the original variables $\{x_{k+1}, \ldots, x_n\}$. Therefore,

$$U_3 = Q_2 + P_2 + S + T$$

It follows that P_1P_2 and Q_1Q_2 are not PI's of F. Contradiction.

Next, consider $Q_2 \subseteq \overline{U_2}$. Arguing in a similar manner as before, we can show

$$U_3 = U_1U_2 + \overline{U}_1\overline{U}_2 + R$$

where R is some function of U_1 and U_2. Further, using the same arguments as in Theorem 4.2.2, we can show U_1U_2 and $\overline{U}_1\overline{U}_2$ are

PI's of U_3. That is, U_3 is not unate in U_1. Contradiction.

Therefore, the conclusion is that if U_3 is unate in U_1, then the sum of the terms under A_1 cannot be equal to 1.

Proof of converse: Let $\sum_i P_1^i \neq 1$ and assume U_3 is not unate in U_1 or $\overline{U_1}$. We shall show this assumption to be contradictory. According to assumption, both U_1 and $\overline{U_1}$ occur in the PI's of U_3. WLOG, let P_1P_2 be a PI of F such that $P_1 \subseteq U_1$ and $P_2 \subseteq U_2$. Let Q_1Q_2 be another PI of F such that $Q_1 \subseteq \overline{U_1}$. That $Q_1 \subseteq \overline{U_1}$ should be present follows from the assumption U_3 is not unate in U_1 or $\overline{U_1}$. For, if $\sum_i P_1^i = U_1$, then using Theorem 4.2.2, we could write

$$U_3 = U_1 * T$$

where * is one member of the set $\{\cdot, +\}$ and T is an expression free of U_1. We could thus show U_3 to be unate in U_1, contrary to original assumption.

By the Corollary to Theorem 4.2.2, the PI's necessary to define U_1 and $\overline{U_1}$ must be present under A_1. The sum of these PI's will be equal to $(U_1 + \overline{U_1})$ or 1. This contradicts the given condition $\sum_i P_1^i \neq 1$. Hence U_3 is unate if $\sum_i P_1^i \neq 1$. Q.E.D.

Theorem 4.2.5 If a function contains two PI's of the form P_1P_2 and Q_1 (or Q_2), then it is not realizable.

Proof: Suppose there exist two PI's like P_1P_2 and Q_1 in F. Assume, WLOG, that $P_1 \subseteq U_1$. If possible, let Q_1 also $\subseteq U_1$. When Q_1 is true, $U_1 = 1$ and irrespective of what U_2 is, $F = 1$. Therefore, $F = U_1 + T$ where T is some expression involving U_1 and U_2 in general. But

$$U_1 = P_1 + Q_1 + S \text{ (by assumption)}$$

where S is some expression involving the original variables. Therefore

$$F = P_1 + Q_1 + S + T$$

That is, P_1P_2 is not a PI of F. Contradiction.

Hence, P_1 and Q_1 must belong to complementary functions. So let $Q_1 \subseteq \overline{U_1}$. Also, WLOG, let $P_2 \subseteq U_2$. When P_1P_2 is true, we have

$$U_1 = 1,\ U_2 = 1, \text{ and } U_3 = 1. \quad \ldots\ldots\ldots\ldots \quad (1)$$

When Q_1 is true so that $U_1 = 0$ and input conditions are such that $U_2 = 1$, then we have, since Q_1 is a PI of F

$$U_1 = 0,\ U_2 = 1, \text{ and } U_3 = 1. \quad \ldots\ldots\ldots\ldots \quad (2)$$

Therefore, $F = U_3(U_1, U_2)$

$= U_2 + R$ . . . . . .from (1) and (2)

where R is some expression involving U_1 and U_2 in general. But

$$U_2 = P_2 + Q$$

where Q is some expression involving the original variables $\{ x_{k+1}, \ldots, x_n \}$. Therefore,

$$F = P_2 + Q + R \ .$$

That is, $P_1 P_2$ is not a PI of F. Contradiction.

Hence, the function cannot be realizable.

Q.E.D.

These theorems establish the basis of a procedure for testing realizability of a function with disjointly decomposed structure. Theorem 4.2.1 states that if we divide the PI's of F in the manner of Table 4.2.1, we get the PI's of the constituent functions U_k and $\overline{U}_k$ for k = 1,2. By the corollary to Theorem 4.2.2, it has been shown that the PI's of U_k $(\overline{U_k})$ obtained in this manner completely define $U_k(\overline{U_k})$. Theorems 4.2.2 and 4.2.4 also clearly indicate how to distinguish a PI of U_k from a PI of $\overline{U_k}$. However, if all PI's of F are not present in the function specification, it may need a little more effort to determine whether a PI under, say A_1, belongs to U_1 or $\overline{U_1}$. The first PI may be assigned arbitrarily to U_1 or $\overline{U_1}$ while the others are to be compared with the previous ones to see if the conditions of Theorem 4.2.2 are directly applicable. Otherwise, using comparison, if a PI Q_1 is seen to have a minterm

common with another PI, already assigned to, say U_1, then $Q_1 \subseteq U_1$ (by the proof of Theorem 4.2.1). As convenient, this comparison may also be done with the second part P_2 of a PI P_1P_2. Let P_1P_2 and Q_1Q_2 be two PI's of F. Then, if it is found that P_2 has a minterm common with Q_2, then assign Q_1 to U_1 if $P_1 \subseteq U_1$. The proof of this statement follows from simple arguments similar to ones given before--to show that, if in this case, $Q_1 \not\subseteq U_1$, then Q_2 and P_2 must be PI's of F, contrary to original assumption.

The theorems are easily seen to apply in cases where the original array is divided into more than two sub-arrays. The generalized theorems are being stated without proof here. The proofs will be found in Appendix B.

Let A be a set of binary variables $\{ x_1, x_2, \ldots, x_n \}$, partitioned into s disjoint subsets $A_1, A_2, \ldots, A_s$. Let $U_1, U_2, \ldots, U_s$ be arbitrary switching functions on the variables of $A_1, A_2, \ldots, A_s$ respectively and U be any switching function on $U_1, U_2, \ldots, U_s$. Let $F(x_1, x_2, \ldots, x_n)$ be a function of n variables, realizable as U.

Theorem 4.2.1' Let $P_aP_b \ldots P_i \ldots P_k$ be a PI of F. Then P_i is a PI of either U_i or $\overline{U}_i$ for i = a,b,..i,..k.

Theorem 4.2.2' Let $P_i X$ and $Q_i Y$ be two PI's of F where X and Y represent products of literals of the variables of any subsets other than A_i. If $X = Y$, and $P_i \subseteq U_i (\overline{U}_i)$, then $Q_i \subseteq U_i (\overline{U}_i)$ and both P_i, Q_i are PI's of $U_i (\overline{U}_i)$. Conversely, let P_i, $Q_i \subseteq U_i (\overline{U}_i)$ and $P_i X \subseteq F$. Then $Q_i X \subseteq F$. Further, if P_i, Q_i are PI's of $U_i(\overline{U}_i)$ and $P_i X$ is a PI of F, then $Q_i X$ is a PI of F.

Corollary 1' The set of PI's of U_i and $\overline{U}_i$ obtained from F with the use of Theorem 4.2.1' completely define U_i and $\overline{U}_i$ for all i.

Theorem 4.2.3' Let $U_1(x_1, \ldots x_k)$, $U_2(x_{k+1} \ldots x_m), \ldots U_s(x_p, \ldots x_n)$ be s Boolean functions defined on disjoint subsets of variables. If $U_1, U_2, \ldots U_s$ are expressed as a sum of prime implicants, such as $U_i = (P_i + Q_i + R_i + \ldots)$, then each resultant product term when

$$F(x_1, \ldots, x_k) = U_1 U_2, \ldots U_s$$

is expressed as a sum of products in a prime implicant of F.

Theorem 4.2.4' U is unate in U_i or $\overline{U}_i$ for any i, if and only if the sum $\sum_j P_i^j$ of the terms of the variables of A_i obtained from F is not equal to 1.

Theorem 4.2.5' If a function contains two PI's of the form $P_i X$ and Q_i for any i, where X is free of the literals of the variables of A_i,

then the function is not realizable as U.

Types of Decomposable Functions

Consider Figure 4.2.1. A given function may include the following two types of products shown in Table 4.2.2.

Table 4.2.2 Types of Prime Implicants in Two-input ULM case

Type I - P_1, P_2

Type II - Q_1Q_2

According to Theorem 4.2.5, a realizable function may contain (a) only type I or (b) only type II products. For case (a), a network with OR-gate at the final level is necessary. The first-level logic functions U_1 and U_2 are determined by the set of products present under A_1 and A_2. Since a ULM is assumed with inputs fed by the variables of each set, hence these functions can always be realized. Case (b) is slightly more complex. Here each product Q_1 must be tested to see if it belongs to U_1 or $\bar{U}_1$. The same is true for Q_2. While this is being done, a truth table of the variables U_1 and U_2 may be filled up to determine the function at the final level. In the first level, the functions U_1, U_2 can be realized or their complementaries with suitable modification at the second level. We shall work out an

example below for illustration.

Example 4.2.1:

$F = x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_3 + \bar{x}_1x_4 + \bar{x}_2x_3 + \bar{x}_2x_4$ and $A_1 = \{x_1, x_2\}$,

$A_2 = \{x_3, x_4\}$

Step 1: Make a table (shown in Table 4.2.3). This is an

Table 4.2.3 Decomposition Table for $F = x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_3 + \bar{x}_1x_4 + \bar{x}_2x_3 + \bar{x}_2x_4$

$A_1 = \{x_1, x_2\}$	$A_2 = \{x_3, x_4\}$
$\bar{x}_1$	x_3
$\bar{x}_1$	x_4
$\bar{x}_2$	x_3
$\bar{x}_2$	x_4
x_1x_2	$\bar{x}_3\bar{x}_4$

example of case (b).

Step 2. Analyze.

Applying Theorem 4.2.4, U_3 is non-unate.

Let $\bar{x}_1 \subseteq U_1$. Applying Theorem 4.2.2,

$\bar{x}_2 \subseteq U_1$. Applying Theorem 4.2.4, $x_1x_2 \subseteq \bar{U}_1$

Similarly $x_3 \subseteq U_2$

$$x_4 \subseteq U_2$$

$$\bar{x}_3 \bar{x}_4 \subseteq \overline{U_2}$$

Step 3: Make a truth table (Table 4.2.4).

Table 4.2.4 A Truth Table for U_3 in terms of Intermediate Level Functions U_1, U_2 in Example 4.2.1

U_1	U_2	U_3
0	0	1
0	1	0
1	0	0
1	1	1

Step 4: Synthesize.

$$U_3 = U_1 \circledcirc U_2$$

$$U_1 = \bar{x}_1 + \bar{x}_2$$

$$U_2 = x_3 + x_4$$

It may be pointed out that the analysis is done in a general manner and constitutes a sufficient test for realizability.

The case of larger (more inputs) ULM's in the intermediate levels and the case of multilevel networks may now be considered. First, let us take a two-level, three-input ULM (intermediate level) network (Figure 4.2.2). Following previous discussion, consider the type of products that may be present in a sum-of-PI form of expression for the function. These are listed in Table 4.2.5.

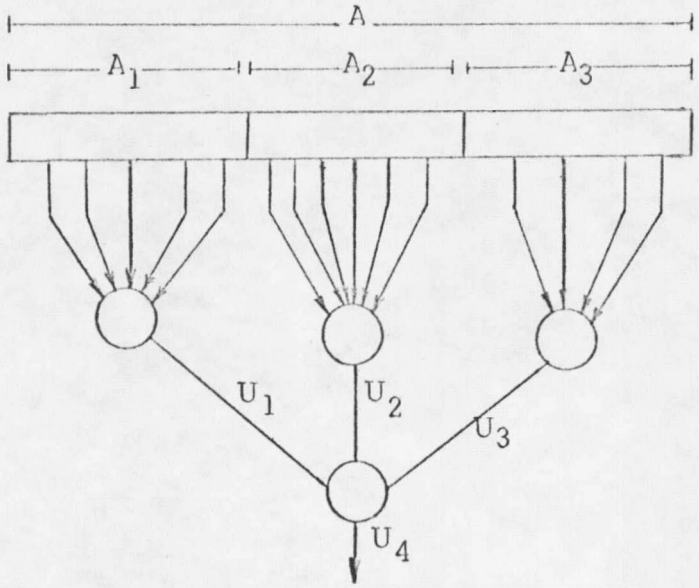


Figure 4.2.2 Two-level Network with Three-input ULM at the Last Level

Table 4.2.5 Types of Prime Implicants in Three-input ULM Case

Member	A	B	C
Type I --	P_1	P_2	P_3
Type II --	Q_1Q_2	R_2R_3	S_1S_3
Type III --	$T_1T_2T_3$		

It is noted that there are $\binom{3}{1}$ members of type I, $\binom{3}{2}$ members of type II and $\binom{3}{3}$ members of type III. There are seven ways in which these types may combine and occur in a function. All these combinations do not represent realizable functions. Consider a function F containing type I PI's. If P_i is in F, then higher type (II and III) PI's involving product terms of the form Q_i cannot be in F (by Theorem 4.2.5'). Therefore, if P_1 occurs as a PI in F, then only R_2R_3 can occur as a PI of F. If two members of type I are in F, then type II and type III cannot occur. All members of type I can also occur with the same restriction. When type I PI's are absent, only type II PI's or type III PI's or under special conditions, type II PI's with type III PI's can be present in F.

It should be clear that permissible combination of types may not give a realizable function, and for testing realizability, Theorem 4.2.2'

must ultimately be used. Still larger ULM network may be analyzed in a similar manner.

A larger number of levels does not involve more complexity than is associated with computing an increased number of intermediate functions due to an increased number of levels. For convenience in dealing with multilevel networks, a modification in notation is introduced here. We shall let U_{ij} represent a general ULM in the network. The first subscript i in U_{ij} refers to the i-th level in the network, while the second subscript j to the j-th position in that level. It has been seen that to determine what the function U_{ij} should be, the function produced at the $(i+j)$-th level must be known. Thus the procedure should be to begin at the final level and work backward after computing the functions of the preceding level. A general algorithm based on the previous discussions may now be given.

General Algorithm

An assumption which underlies the discussion here is that, except for the functions which each ULM must perform in order to realize a function, other parameters like the number of sub-arrays, the number of levels and the number of inputs to an ULM are all predetermined. Though the algorithm is applicable for an assumed network structure,

its simplicity permits testing a large number of varieties with a small amount of effort.

Let $A_1, A_2, \ldots, A_s$ be the sub-arrays into which the original array A is divided. Let F be a function, expressed as a sum of prime implicants for which a realizability test is to be performed.

Step 1: Arrange the PI's in a matrix with columns headed by $A_1, A_2, \ldots, A_s$ and each row representing one PI. The entry in the (i,j)-th position of the matrix is a product of literals of a set of variables such that the product of literals is a part of the PI represented by the i-th row and the variables to which the literals belong are in the sub-array A_j. The PI's are entered in groups according to ascending order of type numbers.

Step 2: Consider the ULM at the last level. Corresponding to each of its inputs, group the sub-arrays (by tracing back from the output to the input array) which produce this input. Call this group of sub-arrays as I-set.

Step 3: Examine the terms of an I-set and determine if the output function is unate in the variable represented by the I-set. (Test procedure becomes simpler if it is found to be unate). If the I-set is

too large and determination of unateness is too involved, then proceed without determining unateness.

Assign the first term of an I-set to a function. Compare the succeeding terms with the previous terms as discussed before and assign them to the function or to its complementary in accordance with Theorem 4.2.2'. If the output function is unate in the input variable represented by the I-set, then all the terms of the I-set must together belong to the same function (preceding level). If a term cannot be assigned first, keep it in a waiting list for that I-set and compare after a sufficient number of terms have been assigned. (If there appears no indication for assigning it to the function or the complementary, as discussed previously, then, the function is not realizable)

Make a check that the function and its complementary are determined completely such that each term satisfies the conditions of Theorem 4.2.2'. In order to check, it is necessary to see, along with the conditions of Theorem 4.2.2', that each PI of a function has at least one literal conflicting with every PI of its complementary, when present, and that the sum of the PI's of these two functions equals 1.

Step 4: Write the functions performed by the preceding level ULM's as sums of PI's obtained from step 3. (Either a function or its complementary can be realized by a ULM.)

Determine the function performed by the last level ULM in terms of the outputs of the preceding level ULM's by writing a set of true conditions in terms of these outputs. Strike out the terms of the complementary functions (preceding level) from the I-sets.

Step 5: Take the functions in the I-sets and, considering each preceding level ULM as the last ULM, follow steps 2 - 4, and continue until functions performed by all ULM's are determined.

Example 4.2.2

$$F = \bar{x}_1x_3x_4x_7x_8 + \bar{x}_1\bar{x}_3\bar{x}_4x_7x_8 + \bar{x}_2x_3x_4x_7x_8$$
$$+ \bar{x}_2\bar{x}_3\bar{x}_4x_7x_8 + \bar{x}_1x_3x_4x_9x_{10}x_{11}$$
$$+ \bar{x}_1\bar{x}_3\bar{x}_4x_9x_{10}x_{11} + \bar{x}_2x_3x_4x_9x_{10}x_{11}$$
$$+ \bar{x}_2\bar{x}_3\bar{x}_4x_9x_{10}x_{11} + x_3\bar{x}_4\bar{x}_5\bar{x}_6x_7x_8$$
$$+ \bar{x}_3x_4\bar{x}_5\bar{x}_6x_7x_8 + x_3\bar{x}_4\bar{x}_5\bar{x}_6x_9x_{10}x_{11}$$
$$+ \bar{x}_3x_4\bar{x}_5\bar{x}_6x_9x_{10}x_{11}$$
$$+ x_1x_2x_5x_7x_8 + x_1x_2x_6x_7x_8 + x_1x_2x_5x_9x_{10}x_{11}$$
$$+ x_1x_2x_6x_9x_{10}x_{11}$$

The network we shall test for realizing this function is shown in Figure 4.2.3.

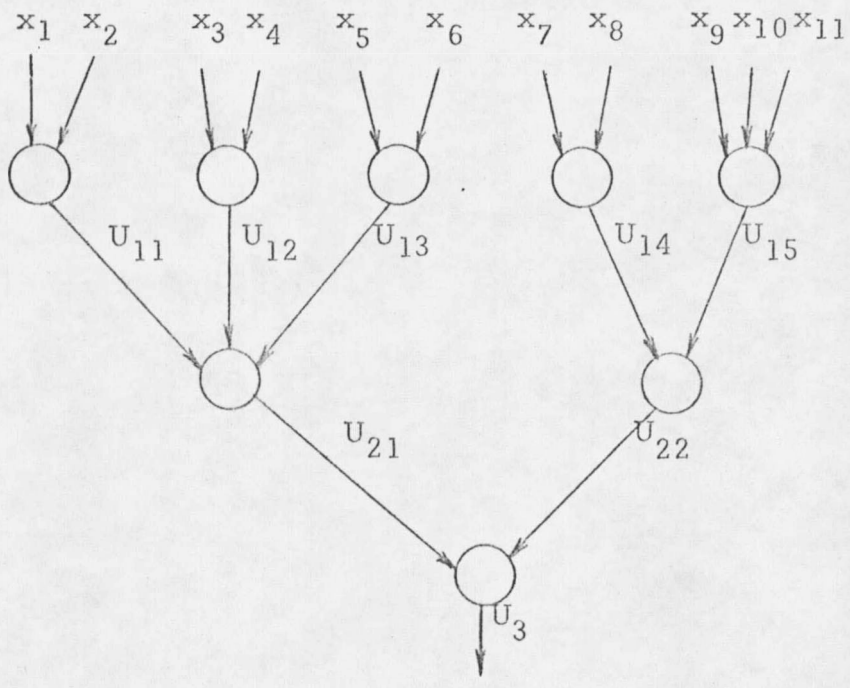


Figure 4.2.3 The Structure of the Network Used for Test-realization of the Function of Example 4.2.2

Step 1: A Table is made.

Step 2: The subarrays corresponding to each I-set are grouped.

The results of these steps are shown in Table 4.2.6.

Table 4.2.6 A Decomposition Table for the Function F in Example 4.2.2

$I_1=\{x_1,\ldots,x_6\}$			$I_2=\{x_7,\ldots,x_{11}\}$	
$A_1=\{x_1,x_2\}$	$A_2=\{x_3,x_4\}$	$A_3=\{x_5,x_6\}$	$A_4=\{x_7,x_8\}$	$A_5=\{x_9,x_{10},x_{11}\}$
$\bar{x}_1$	x_3x_4		x_7x_8	
$\bar{x}_1$	$\bar{x}_3\bar{x}_4$		x_7x_8	
$\bar{x}_2$	x_3x_4		x_7x_8	
$\bar{x}_2$	$\bar{x}_3\bar{x}_4$		x_7x_8	
x_1x_2		x_5	x_7x_8	
x_1x_2		x_6	x_7x_8	
$\bar{x}_1$	x_3x_4			$x_9x_{10}x_{11}$
$\bar{x}_1$	$\bar{x}_3\bar{x}_4$			$x_9x_{10}x_{11}$
$\bar{x}_2$	x_3x_4			$x_9x_{10}x_{11}$
$\bar{x}_2$	$\bar{x}_3\bar{x}_4$			$x_9x_{10}x_{11}$
	$x_3\bar{x}_4$	$\bar{x}_5\bar{x}_6$	x_7x_8	
	$\bar{x}_3x_4$	$\bar{x}_5\bar{x}_6$	x_7x_8	
	$x_3\bar{x}_4$	$\bar{x}_5\bar{x}_6$		$x_9x_{10}x_{11}$
	$\bar{x}_3x_4$	$\bar{x}_5\bar{x}_6$		$x_9x_{10}x_{11}$
x_1x_2		x_5		$x_9x_{10}x_{11}$
x_1x_2		x_6		$x_9x_{10}x_{11}$

Step 3: U_3 is unate in U_{22} and U_{21}. (It may appear difficult to check by addition; but it can be concluded that U_3 is unate in U_{21} because U_3 is a function of two variables and is unate in U_{22}.)

Let $\bar{x}_1 x_3 x_4 \subseteq U_{21}$ and $x_7 x_8 \subseteq U_{22}$; then, $U_3 = U_{21} U_{22}$. A check is made to see this relation is true for all PI's of U_{21} and U_{22}.

Step 4: $U_{21} = \bar{x}_1 x_3 x_4 + \bar{x}_1 \bar{x}_3 \bar{x}_4 + \bar{x}_2 x_3 x_4 + \bar{x}_2 \bar{x}_3 \bar{x}_4 + x_1 x_2 x_5$
$+ x_1 x_2 x_6 + x_3 \bar{x}_4 \bar{x}_5 \bar{x}_6 + \bar{x}_3 x_4 \bar{x}_5 \bar{x}_6$

$U_{22} = x_7 x_8 + x_9 x_{10} x_{11}$

Step 5: To determine U_{21} in terms of U_{11}, U_{12}, U_{13} and U_{22} in terms of U_{14} and U_{15}.

U_{21} is not unate in U_{11}^*, U_{12}^*, and U_{13}^* (where by U^* both U and $\bar{U}$ are implied).

Let $\bar{x}_1 \subseteq U_{11}$. Then $\bar{x}_2 \subseteq U_{11}$. Further $x_1 x_2 \subseteq \bar{U}_{11}$ as $x_1 x_2 x_3 x_4 \not\subseteq U_{21}$. Therefore, $U_{11} = \bar{x}_1 + \bar{x}_2$

Let $x_3 x_4 \subseteq U_{12}$. Then $\bar{x}_3 \bar{x}_4 \subseteq U_{12}$. Further $x_3 \bar{x}_4 \subseteq \bar{U}_{12}$ as $\bar{x}_1 x_3 \bar{x}_4 \not\subseteq U_{21}$. Also $\bar{x}_3 x_4 \subseteq \bar{U}_{12}$. Therefore $U_{12} = x_3 \oplus x_4$.

Let $x_5 \subseteq U_{13}$. Then $x_6 \subseteq U_{13}$ and $\bar{x}_5 \bar{x}_6 \subseteq \bar{U}_{13}$.

Therefore $U_{13} = x_5 + x_6$.

From Table 4.2.6, we have

if $U_{11} = 1$, $U_{12} = 1$, then $U_{21} = 1$

if $U_{11} = 0$, $U_{13} = 1$, then $U_{21} = 1$ and

if $U_{12} = 0$, $U_{13} = 0$, then $U_{21} = 1$.

Therefore

$$U_{21} = U_{11}U_{12} + \bar{U}_{12}\bar{U}_{13} + \bar{U}_{11}U_{13}$$

A check is made at this stage to see if this relation is true for all the PI's of U_{11}, U_{12} and U_{13}. Next determine U_{22} in terms of U_{14} and U_{15}.

U_{22} is unate in U_{14}* and U_{15}*. Let $x_7x_8 \subseteq U_{14}$ and

$x_9x_{10}x_{11} \subseteq U_{15}$. Then

$U_{22} = U_{14} + U_{15}$

$U_{14} = x_7x_8$

$U_{15} = x_9x_{10}x_{11}$

Figure 4.2.4 shows the functions of the different ULM's in the network.

A simple example of a non-realizable function will be worked out below.

Example 4.2.3

$$F = x_1x_2x_6 + x_1x_4 + x_2x_5 + x_3x_6$$

and $A_1 = \{x_1, x_2, x_3\}$, $A_2 = \{x_4, x_5, x_6\}$

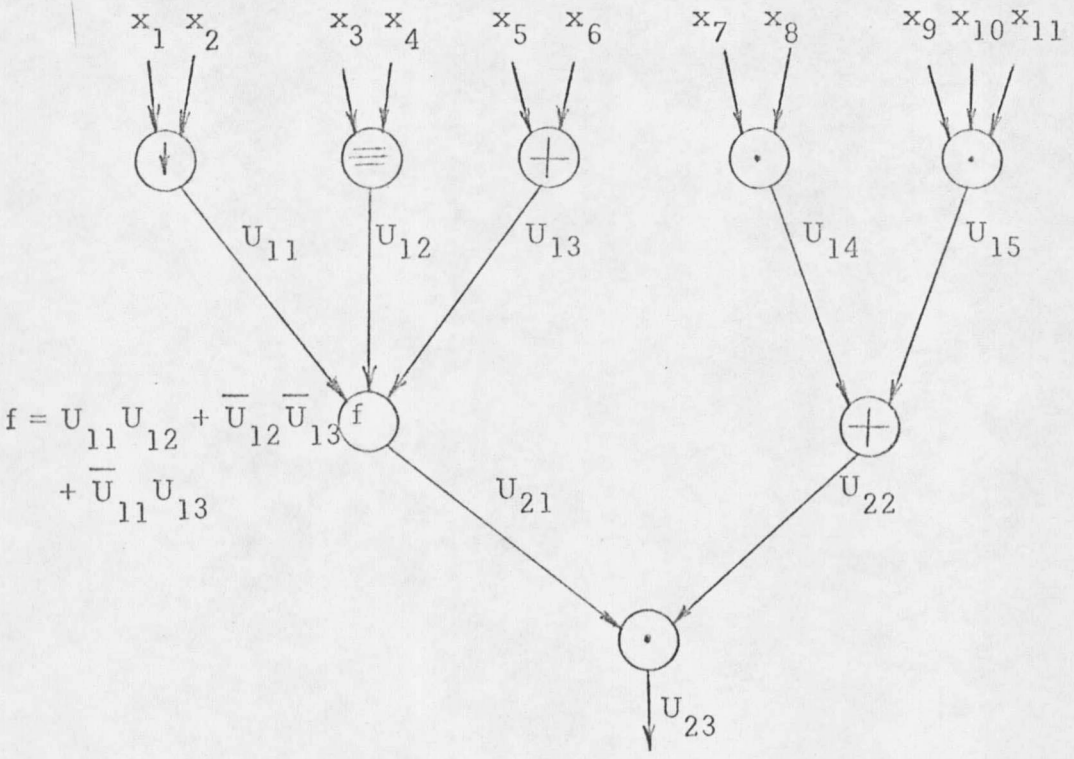


Figure 4.2.4 Specification of the Network to Realize the Function of Example 4.2.2

Step 1: As before, a table is made (Table 4.2.7)

Table 4.2.7 A Decomposition Table for $F = x_1x_2x_6 + x_1x_4 + x_2x_5 + x_3x_6$

$A_1 = \{ x_1, x_2, x_3 \}$	$A_2 = \{ x_4, x_5, x_6 \}$
x_1	x_4
x_2	x_5
x_3	x_6
x_1x_2	x_6

Step 2: Analysis: Let $x_1 \subseteq U_1$

Then $x_2 \subseteq \overline{U}_1$ as $x_2x_4 \not\subseteq F$.

But $x_1x_2 \subset x_1 \subseteq U_1$ and $x_1x_2 \subset x_2 \subseteq \overline{U}_1$. This is impossible. Hence the function cannot be realizable.

A point that may be noticed in this particular example is that the product terms occurring under A_1 are not all PI's ($\because x_1x_2 \subset x_1$). Thus the function does not satisfy the conditions of Theorem 4.2.1.

4.3 Non-disjoint Decomposition

The decomposition of an array into non-disjoint sub-arrays can be expected to permit a wider class of logical functions to be realized, because, if the idea is stretched to the extreme, the result may be a trivial decomposition of an array into a number of sub-arrays,

each sub-array being equal to the original array so that there will be no problem in realizing arbitrary functions on the array by realizing it on the sub-arrays. In this section, some simple cases of non-disjoint decomposition will be discussed.

Let $A_1, A_2, \ldots A_s$ be a linear arrangement of sub-arrays such that each adjacent pair of sub-arrays -- A_i and A_{i+1} -- have some variables common between them (Figure 4.3.1). A_1 and A_s have each only one neighboring sub-array A_2 and A_{s-1} respectively, while all other sub-arrays have two neighbors. Let $D_1, D_2, \ldots D_i, \ldots D_s$ be the subsets of $A_1, A_2, \ldots A_i, \ldots A_s$ respectively such that D_i contains all those variables of A_i that belong only to A_i and to no other sub-arrays. Let $C_1, C_2, \ldots C_{s-1}$ be each a set of variables such that C_i contains all those variables that are common to A_i and A_{i+1} for $i=1,2,\ldots,(s-1)$.

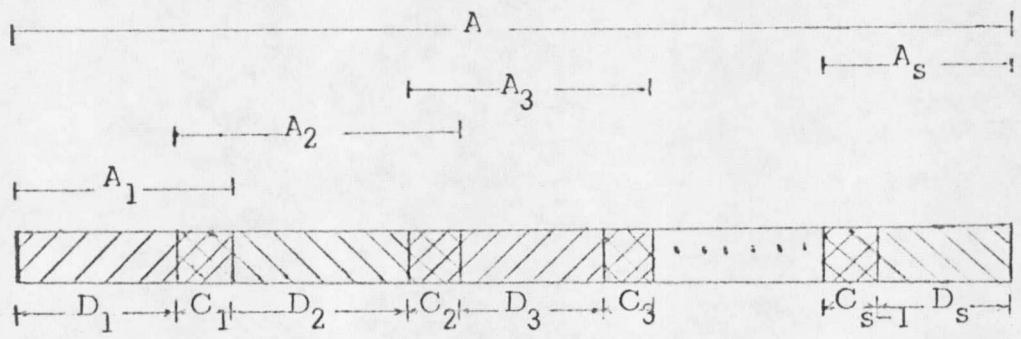


Figure 4.3.1 Linearly Arranged Non-disjoint Sub-arrays

The case to be considered is when A is composed of two sub-arrays--A_1 and A_2 (figure 4.3.2). The possible types of PI's that may be present in a given function are listed under Table 4.3.1.

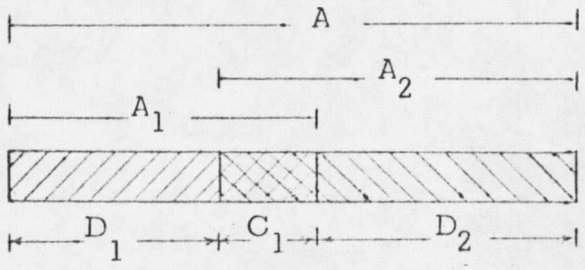


Figure 4.3.2 Decomposition into Two Sub-arrays

Table 4.3.1 Types of Prime Implicants in Simple Non-disjunctive Case

Types	Members
I	P_1, P_2
II	Q_1X_1, X_1Q_2
III	R_1R_2
IV	$S_1Y_1S_2$
V	Z_1

As before, P_1, Q_1, R_1, etc. -- letters (excepting X, Y, Z) with subscript 1 -- are used to represent products of the literals of the variables of D_1; similarly, P_2, Q_2, R_2 etc. for the variables of D_2;

X_1, Y_1, Z_1 for the variables of C_1.

The combination of different types of PI's of a function can be studied for realizability in accordance with the previous discussion and a synthesis procedure can also be worked out in a similar fashion. Most of the theorems apply with slight modification. These are stated without proof. The proofs are given in Appendix B. The X's and Y's below may be vacuous.

Theorem 4.3.1 Let $P_1X_1P_2$ be a PI of F. Then P_1X_1 is either a PI of U_1 or $\overline{U}_1$ and X_1P_2 is either a PI of U_2 or $\overline{U}_2$.

Theorem 4.3.2 If $P_1X_1P_2$ and $Q_1Y_1Q_2$ are two PI's of F and if $X_1P_2 = Y_1Q_2$, then $Q_1Y_1 \subseteq U_1(\overline{U}_1)$ if $P_1X_1 \subseteq U_1(\overline{U}_1)$ and further P_1X_1 and Q_1Y_1 are PI's of $U_1(\overline{U}_1)$. Conversely, if P_1X_1, $Q_1X_1 \subseteq U_1(\overline{U}_1)$, then $Q_1X_1P_2 \subseteq F$ if $P_1X_1P_2 \subseteq F$. Further, if P_1X_1 and Q_1X_1 are PI's of $U_1(\overline{U}_1)$ and $P_1X_1P_2$ is a PI of F, then $Q_1X_1P_2$ is a PI of F.

As the sub-arrays are not disjoint, some terms in a given function may seem eligible for inclusion in more than one constituent functions U_1, U_2, etc. But not all the alternatives may ultimately lead to realization. Example 4.3.2 is an illustration of this case. The phrase 'the term occurring under A_1 (A_2)' will mean here that if

$P_1X_1P_2$ is a PI of F, then P_1X_1 occurs under A_1 and X_1P_2 occurs under A_2. Then, corollary to Th. 4.2.2 applies in non-disjoint case. But, theorem 4.2.3 is not applicable. However, theorems 4.2.4 and 4.2.5 apply as such. The theorems can also be generalized in the same manner as in the disjoint case (shown in Appendix B).

For a function to be decomposable in the manner of Figure 4.3.2, one of several cases may occur (with ref. to Table 4.3.1).

Case 1. The function consists of only type I or type II or type V PI's or PI's of any combination of these types. This case is similar to type I of Table 4.2.2 and the synthesis procedure same. Type I PI must not occur with any other type (by Th. 4.2.5).

Case 2. The function consists of type II and type III PI's.

Let $R_1 \subseteq U_1$	and	$R_2 \subseteq U_2$	Then, either
$Q_1X_1 \subseteq U_1$	and	$X_1 \subseteq U_2$	or
$Q_1X_1 \subseteq \overline{U}_1$	and	$X_1 \subseteq \overline{U}_2$	

Also, Theorem 4.3.2 must be satisfied for ensuring realizability.

A function with other combinations of Types II, III, IV and V PI's can be similarly analyzed. It is obvious that as the two sub-arrays are made to share some variables, a greater variety of functions

becomes realizable. Networks using three-input ULM's in the intermediate levels may also be similarly studied. The list of different possible types of PI's in this case becomes quite large. Two examples to illustrate the previous discussions are given below.

Example 4.3.1

$$F = \bar{x}_1 x_2 \bar{x}_4 + x_1 \bar{x}_2 \bar{x}_4 + \bar{x}_3 \bar{x}_5 + \bar{x}_3 \bar{x}_6 + x_1 x_2 x_3 + x_1 x_2 x_5 x_6 + \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 \bar{x}_2 x_5 x_6 + x_3 x_4 + x_4 x_5 x_6$$

The network to test for the realization of F is shown in Figure 4.3.3.

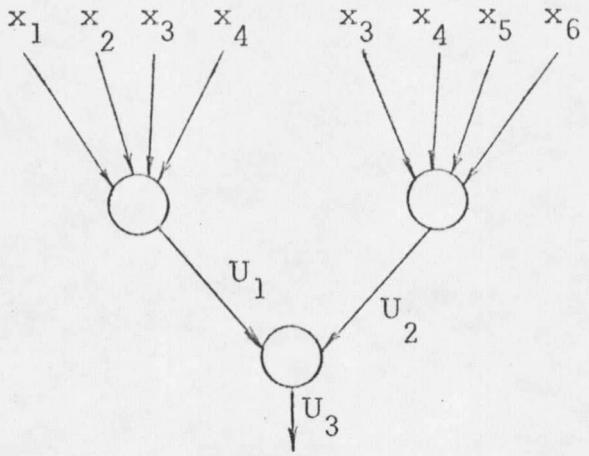


Figure 4.3.3 A Network to Realize F in Example 4.3.1

Step 1: A table is made (Table 4.3.2).

Table 4.3.2 A Decomposition Table for the Function F in Example 4.3.1

A_2

A_1

$D_1 = \{x_1, x_2\}$	$C_1 = \{x_3, x_4\}$	$D_2 = \{x_5, x_6\}$
$x_1 x_2$	x_3	
$x_1 x_2$		$x_5 x_6$
$\bar{x}_1 \bar{x}_2$	x_3	
$\bar{x}_1 \bar{x}_2$		$x_5 x_6$
$x_1 \bar{x}_2$	$\bar{x}_4$	
$\bar{x}_1 x_2$	$\bar{x}_4$	
	$\bar{x}_3$	$\bar{x}_5$
	$\bar{x}_3$	$\bar{x}_6$
	x_4	$x_5 x_6$
	$x_3 x_4$	

Step 2: Analysis. The unateness determination is omitted here.

Let $x_1 x_2 \subseteq U_1$ and $x_5 x_6 \subseteq U_2$

then $\bar{x}_1 \bar{x}_2 \subseteq U_1$ and $x_4 \subseteq U_1$

Further, $x_3 \subseteq U_2$ as $x_5x_6 \subseteq U_2$.

$x_1\bar{x}_2 \subseteq \bar{U}_1$ as $x_1\bar{x}_2x_5x_6 \not\subseteq F$

Therefore, $\bar{x}_1x_2 \subseteq \bar{U}_1$

Therefore, $\bar{x}_4 \subseteq \bar{U}_2$ as F is non-degenerate

$\bar{x}_5 \subseteq \bar{U}_2$ as $x_1x_2\bar{x}_5 \not\subseteq F$

$\bar{x}_3 \subseteq \bar{U}_1$ as F is non-degenerate.

Step 3: We have, $U_1 = x_1x_2 + \bar{x}_1\bar{x}_2 + x_4$; $U_2 = x_3 + x_5x_6$

A truth table of U_3 in terms of U_1 and U_2 is shown (Table 4.3.3).

Table 4.3.3 A Truth Table of U_3 in terms of Intermediate Level Functions U_1 and U_2 in Example 4.3.1

U_1	U_2	U_3
1	1	1
0	0	1
1	0	0
0	1	0

From the table, it is seen that $U_3 = U_1U_2 + \overline{U_1}\ \overline{U_2}$.

Example 4.3.2

$$F = x_1x_4 + x_2x_4 + x_3x_4 + x_5 + x_6x_7 + x_8x_9$$

The network to be tested for realizing the function is shown in Figure 4.3.4.

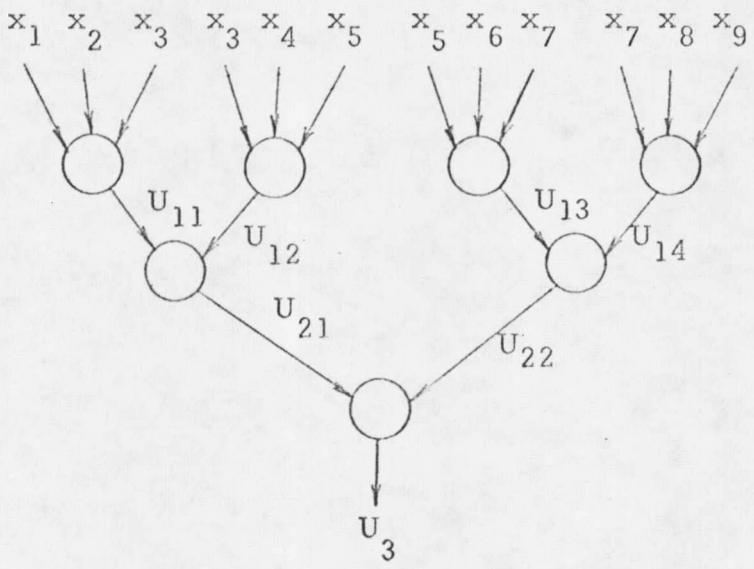
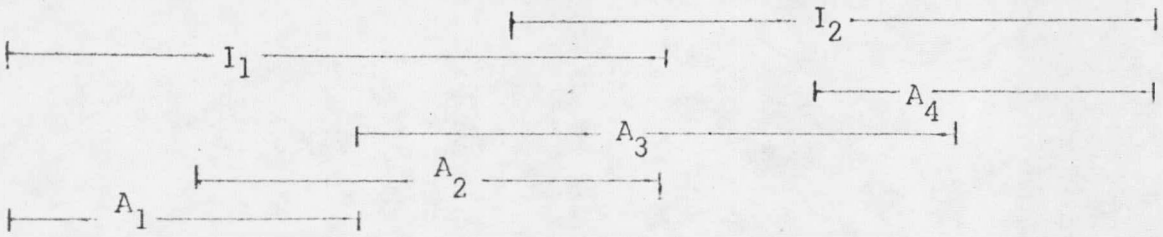


Figure 4.3.4 The Network to Realize F in Example 4.3.2

In the figure, U_{21} is some function $f(x_1 \ldots x_5)$ and U_{22} is another function $g(x_5 \ldots x_9)$. In general, the PI x_5 of F may belong either to U_{21} or to U_{22} or to both. The purpose of this example is to show that x_5 cannot belong to U_{21} and thereby demonstrate the existence of the problem of determining the proper set of terms for the constituent functions in a preceding level such that a given function may be ultimately realized, provided it is realizable by the given network structure.

In the first step a table is made (Table 4.3.4).

Table 4.3.4 A Decomposition Table for $F = x_1x_4 + x_2x_4 + x_3x_4 + x_5 + x_6x_7 + x_8x_9$



$D_1=\{x_1,x_2\}$	$C_1=\{x_3\}$	$D_2=\{x_4\}$	$C_2=\{x_5\}$	$D_3=\{x_6\}$	$C_3=\{x_7\}$	$D_4=\{x_8,x_9\}$
x_1		x_4				
x_2		x_4				
	x_3	x_4				
			x_5			
				x_6	x_7	
						x_8 x_9

Considering the function U_3 in terms of the variables U_{21} and U_{22}, it is seen that U_3 consists of only type I and type V PI's and therefore an OR gate at the final level is all that is necessary. Moreover, U_3 is unate in U_{21} and U_{22} so that we may write

$$U_{21} = x_1x_4 + x_2x_4 + x_3x_4 + x_5$$

$$U_{22} = x_5 + x_6x_7 + x_8x_9$$

As it is not known where x_5 should go, it is included in both the functions. But consider U_{21}. It is a sum of PI's of types I, II, and III. So it is not realizable by the network.

Let $U_{21} = x_1x_4 + x_2x_4 + x_3x_4$. In a manner similar to the previous example, it can be found that $U_{21} = U_{11} \cdot U_{12}$ and

$$U_{11} = x_1 + x_2 + x_3$$

$$U_{12} = x_4 + x_3x_4$$

$$= x_4$$

Also $U_{22} = U_{13} + U_{14}$ and

$$U_{13} = x_5 + x_6x_7$$

$$U_{14} = x_8x_9.$$

Thus it is seen that if x_5 is not included in U_{21}, then the function is realizable by the given network. Mention may be made of the fact that in U_{12} an implicant (not PI) x_3x_4 is obtained. This is unlike the situation in disjoint decomposition.

The algorithm for test-synthesis of non-disjointly arranged network structure is not separately given as all the steps are same as in the algorithm in disjoint case. The possible ambiguity in determining the constituent functions of a preceding level, as illustrated, may be

resolved by first assuming the terms concerned to be present in all the constituent functions for which they are eligible and continuing the analysis until either the given function is found to be realizable or some constituent functions are found to be unrealizable. In the latter case, the removal of suitable terms may be attempted on a cut-and-try basis.

Chapter 5

PARALLEL BULK TRANSFER SYSTEM WITH FLEXIBLE INPUT DOMAIN

5.1 The Problem of Variable Grouping

The algorithm for the test-systhesis of functions realizable with parallel bulk transfer system which has been presented in the previous Chapter is dependent on a predetermined grouping of variables. Such a situation corresponds to a parallel B.T. system where individual B.T. units process a fixed set of variables on the input array. Alternately, it is possible to think of the input domain of a B.T. unit as a flexible set of variables, which may be anywhere on the input array. Consequently, the question of a proper grouping of variables arises in the testing of a given function for realizability. Of course, it is possible to assume one particular grouping at a time and apply the previous algorithm until one works or all possible groupings fail. But it is a laborious process. The purpose of this Chapter is to study some aspects of this problem and develop an alternate procedure for determining the grouping of different variables that may lead to the realization of the function through decomposition.

The decomposition of a function into subfunctions usually consists in factoring out of suitable terms and collection of other terms into proper groups. It is simpler to determine which terms to factor out in the case of disjunctive decomposition than when non-disjunctive

decomposition is also allowed. It is further simplified if a restriction is put on the logic-realizing elements which have so long been assumed to be ULM's so that only unate functions of the inputs are produced. Apart from the fact that the underlying mathematical structure of unate functions can be exploited, this restriction may considerably reduce the complexity of logical module.

Given a switching function of any number of variables, there always exists a large enough ULM that can realize it. However, because of engineering difficulties it is not practicable to produce ULM of large number of variables. Three or four-variable ULM may be taken as standard. Thus, with a given function, the attempt should be to decompose it into sub-functions of small number of variables. In this Chapter, we shall make a study of some specific cases of disjunctive and non-disjunctive decomposition with the help of the properties of realizable functions studied in the previous Chapter. The cases include unate-logic element network and limited-input ULM network.

5.2 Unate Logic Network

Mukhopadhyay[14] has studied properties of unate cascades and suggested minimization algorithms for them. The unate cascades are similar in structure to the Maitra cascade--the distinction being that the cells in the cascade are restricted to perform only unate frunctions of their two inputs. In this section, the more general tree

network of unate logic elements having more than two inputs will be studied. Each logic element is assumed to be capable of producing any unate function of its inputs. It will be called a Unate Module (UM).

Consider the tree network shown in Figure 5.2.1. The function U_{ij} for proper i, j with $1 \leqq i \leqq n$ and $1 \leqq j \leqq m$ has the same meaning as before. With the restriction now that every function U_{ij} is unate, the following result is obtained.

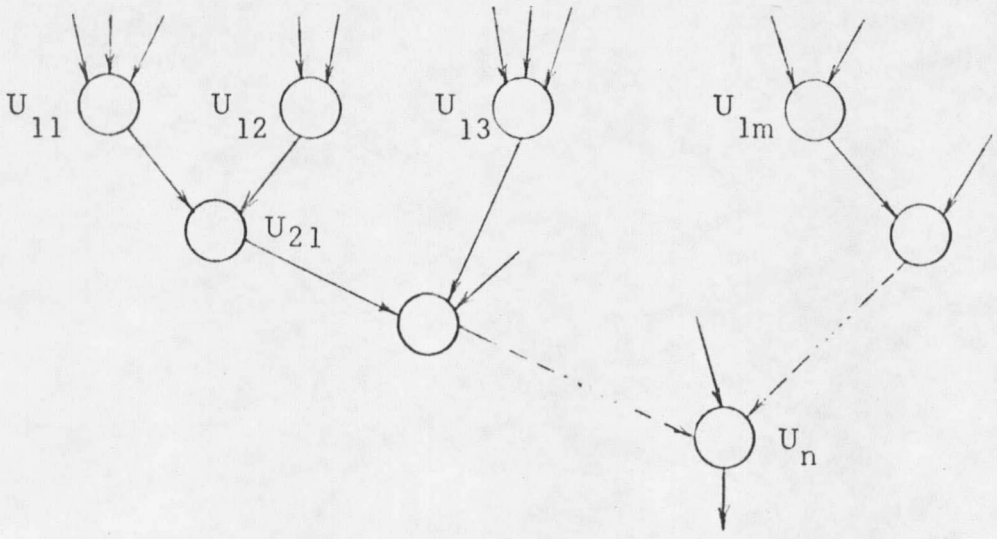


Figure 5.2.1 A Tree Network

Theorem 5.2.1 If the output function produced by every module in the general disjunctive network is unate, then the final output function F is unate in the original variables $\{ x_1, x_2, \ldots x_n \}$.

Proof: Express F in terms of the outputs of the UM's of the last but one stage:

$$F = U_n (U^*_{n-1,1}, U^*_{n-1,2}, \ldots U^*_{n-1,i}, \ldots U^*_{n-1,m})$$

where $U^*_{n-1,i}$ means either $U_{n-1,i}$ or $\overline{U}_{n-1,i}$ but not both. Express $U^*_{n-1,i}$ in terms of the previous level variables for all i. Proceed in this manner back up to the first level. Since the network is disjunctive at all levels, the occurrence of a variable is accounted for by only one UM at any level. As the functions produced are unate, the function F cannot contain a variable in both primed and unprimed form at any stage. Thus F is unate in the original variables.

Q.E.D.

As a result of this property, only unate functions require to be tested for realizability with a unate tree network. For non-unate functions, more than one tree must be used for realization.

Generally there are two types of terms in a function specified as a sum of prime implicants--(i) terms involving only those variables that do not occur in more than one term; (ii) terms involving variables that occur in more than one term. As an example, let

$F = x_1 + x_2 + x_3x_4 + x_5x_6x_7 + x_5x_6x_8$. Here the first three terms belong to type (i) and the last two terms to type (ii). To get an estimate of how many UM's may be necessary to realize a given function, let us restrict consideration to functions which contain only terms of type (i). Given a specific size k of an UM, the number of UM's necessary to realize a prime implicant that is a product of m literals in the manner shown in Figure 5.2.2, is given by

$$N_1 = 1 + \left\lceil \frac{m-k}{k-1} \right\rceil$$

where $\lceil$ represents the ceiling value. If each PI of the function is considered separately, then the total number of UM's required to realize i PI's separately is $N = 1 + \left\lceil \frac{(m_1-k)}{k-1} \right\rceil + 1 + \left\lceil \frac{(m_2-k)}{k-1} \right\rceil + \ldots + \left\lceil \frac{(m_i-k)}{k-1} \right\rceil$

where $m_1, m_2, \ldots m_i$ are the number of literals in the first, second, ..., i-th PI respectively. Further, to realize the function consisting of i prime implicants, it will need $M = 1 + \left\lceil \frac{(i-k)}{k-1} \right\rceil$ UM's to produce a single output. Thus, at most, (N+M) UM's are necessary to realize any given function with k-variable UM's.

Synthesis of Disjunctive Unate Network of k-input Elements

Given a function F, a procedure for test-synthesis with a unate network of k-variable logic elements will be considered here. Of the

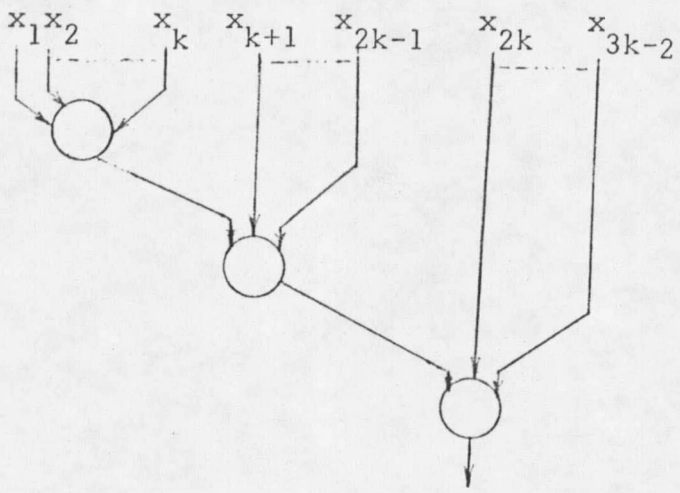


Figure 5.2.2 A Tree with k-Input Logic Elements

two types of prime implicants of a given function discussed earlier in this section, type (i) PI's are already in disjunctively dissociated form as given; it is necessary to realize these terms by grouping in consistence with the size of the logic elements.

As an example, let F = ab + cdef + g. A unate tree of 3-input elements realizing F is given in Figure 5.2.3. For the type (ii) PI's, decomposable features, if any, should be investigated and brought out. For convenience, we define certain terms and prove a property first.

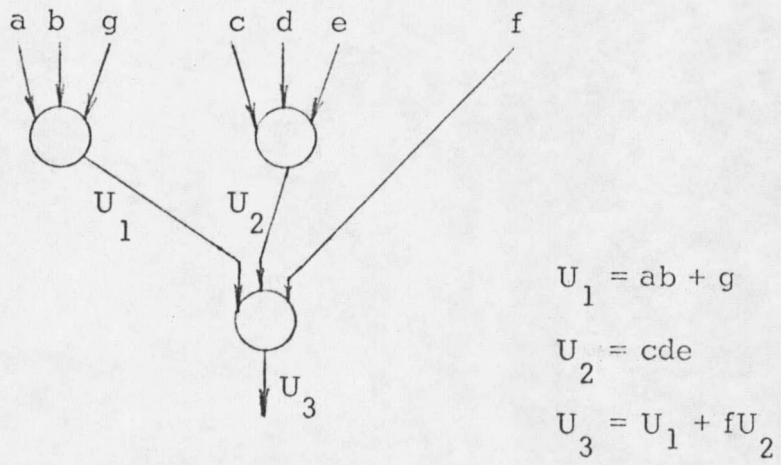


Figure 5.2.3 A Network to Realize F = ab + cdef + g

Definition 5.2.1 A function will be called k-unate decomposable if it can be realized in a tree by using k-input or smaller unate logic elements in a disjunctive manner.

Definition 5.2.2 The length of a PI of a unate function is the number of literals in the PI. The length difference of two PI's of a unate function is the difference in the number of literals contained in the two PI's.

The maximum length of a PI for an n-variable unate function can be n. This occurs when there is only one PI in the function and it consists of the literals of all the variables of the functions. The maximum possible length difference of an n-variable unate function is (n-2). This

occurs when there are two PI's in the function with one PI consisting of only one literal and the other consisting of the remaining (n-1) literals.

Definition 5.2.3 If $F(x_1, x_2, \ldots x_n)$ can be expressed as a composition involving $U_k(x_i, \ldots x_j)$ as an argument such that $\{ x_i, \ldots x_j \}$ is a subset of $\{ x_1, \ldots x_n \}$, then $U_k(x_i \ldots x_j)$ is a sub-function of F. Two functions are disjoint if they are defined on disjoint sets of variables.

Lemma 5.2.1 Let F be k-unate decomposable. Let xA be a PI of F, whose literals occur in at least two disjoint subfunctions of F, where A represents a product of literals. Let U_1 be a subfunction of at most k variables of F containing the prime implicant xB where $B \subseteq A$. If U_1 has any other PI, then there is at least another PI of F, which will contain the literals of A or a subset of these literals such that at most (k-2) of them are absent.

Proof: With the given condition, let B be vacuous in xB. In this case, x is a PI of U_1. By Theorem 4.2.2', the factor A must be associated with every PI of U_1. Thus F will contain other PI's having the factor A. In the other extreme, B can be a product of (k-2) literals. In that case, there exists one other PI of U_1 consisting of a single literal

only. By Theorem 4.2.2', this PI of U_1 must be associated with a factor containing all the literals of A but those of B. Thus F will contain another PI having a factor which will contain all but (k-2) literals of A. When the size of B lies between these two extremes, the common factor correspondingly is of intermediate size. Q.E.D.

The importance of this theorem is in connection with determining a sub-function of a given function, containing a particular variable. To do this, it is not necessary to consider all terms of the given function but only those that have a factor of proper length common. This property along with the fact that the function must be unate makes test-synthesis of unate networks simpler than the general disjunctive network synthesis. In the next section, an algorithm for general disjunctive network synthesis will be given, indicating the special aspects for the unate network at proper places.

5.3 An Algorithm for General Disjunctive Network Synthesis

The tree network given in Figure 5.2.1 may be thought of as a general disjunctive network of universal logical modules. In studying a synthesis procedure for it, we define at first a term applicable here.

Definition 5.3.1 A function is called k-decomposable if it can be realized in a tree by using k-variable ULM's in a disjunctive manner.

Assume that a function is specified as a complete sum of prime implicants. This will be always the case if the function is unate. To test it for k-decomposability, the following steps are developed.

Step 1. Test if the function is unate. If unate then use simplification as mentioned along with the following steps.

Step 2. Inspect and enter in a sub-function all such terms as containing variables that occur in more than one term. Other terms of F consist of only those variables that occur only once. These are already in disjunctively dissociated form and may be realized in consistence with the size of the logic device. From the sub-function obtained above, factor out any common literals and get the residual function.

Step 3. Consider the set of variables in the sub-function separated (or the residual function as the case may be). Take a variable and list it with all the other variables which associate with it in different terms. Take each of the latter variables and add to the list the new variables that associate with them and continue in this manner. Refer

to this list as the list of "connected" variables. If at any time it is found that the list does not grow beyond k variables, then there exists a disjunctive sub-function of at most k variables, whose terms can be separated out. Replace this sub-function by a single variable U_i. Take a new variable not included in the previous list and proceed in the same manner until all the variables are exhausted. Get the residual function after separating all the sub-functions.

For example, let F = ab + bc + ca + de + df. Testing F for 3-decomposability, it is seen that the first term ab involves variables a and b which are associated with the terms ca and ab respectively. These three variables are not associated with any other new variables. Therefore, a sub-function U_1 = ab + bc + ca can be formed.

Step 4. In step 3, if there is a list containing more than k variables take a term of F containing some of these variables. Let P_1 be such a PI and x a literal occurring in P_1. To determine which sub-function contains x, the first step is to break P_1 into two parts: $P_1 = x \mid H$, where H is some product of literals not involving x. Compare H with other PI's to find the common literals in each case. A PI having larger number of common literals may be considered first. Let P_i be a PI of the given function which can be written as $P_i = G \mid I$

where G and I are products of literals with I common to both P_i and H. If the function is to be tested for k-unate decomposability, P_i should be chosen such that it contains a factor having all but at most (k-2) literals of H. Now, H can be written as H = IJ where J is either vacuous or a product of some literals. It is then possible to express $(P_1 + P_i)$ as $P_1 + P_i = x\,I\,J + G\,I = (x\,J + G)\,I$. If xJ and G can be grouped under the same sub-function, then for all terms involving xJ, there must be corresponding terms involving G, satisfying Theorem 4.2.2'. Further, it implies that an implicant of the residual function (xJ + G), which is not a prime implicant, cannot be present.

If these conditions are not satisfied, then G cannot be a term of the sub-function containing xJ. A new term of F must be considered then. If there is no term satisfying the conditions, there may be two possibilities--(i) the sub-function containing x has only one PI or (ii) there is no sub-function of proper size including the variable x. Case (i) may be tested by factoring out the common product from all the terms containing x and defining a function U_j for the common product. The occurrence of the variables of U_j must be replaceable with U_j or $\overline{U}_j$ with suitable arrangement of terms. If this is not possible, then the second possibility may be accepted. If there is only one term in F

containing $x(\bar{x})$, then a term containing $\bar{x}(x)$ may be taken instead of the term containing $x(\bar{x})$, and the step 4 must be restarted. If there is one term containing x and one term containing $\bar{x}$, then the terms containing x and $\bar{x}$ should form a 2-variable sub-function for decomposability. The case when there is only one term containing $x(\bar{x})$ and no term containing $\bar{x}(x)$ is unrealizable except in case that term is the only term in the residual function at that stage. The case (ii) does not mean that F is not disjunctively decomposable. It may indicate that the input x occurs in a higher level of the tree. The next step should then be to choose another variable, say y, and go through the above procedure.

If conditions as mentioned earlier are satisfied for G, then G can be grouped with xJ in a partially-formed sub-function. In this case, move to step 5.

Step 5. Let P_j be a PI which has some literals common with I such that it is possible to write I = ML and P_j = NL. In this case, take out L and write $P_1 + P_i + P_j = L(xJM + GM + N)$. Compare GM with N to ensure, as before, that both can be included in the same sub-function. It is not necessary to compare N with the other term xJM. If it is possible to include N, augment the partially-formed

function accordingly. Take a new term and follow the same procedure as in Step 5. If it is not possible to include N, take another term of F and follow Step 5. When there are no more terms of F to be compared, let T be the common factor of the set of terms which are suitable for inclusion in the function. Replace the residual function obtained by factoring out T by a single variable U_k. It must be possible to replace all occurrences of the variables of U_k by U_k or $\overline{U}_k$. If not, the function is not disjunctively decomposable.

When the sub-function is formed, a check must be made to see if the number of variables has exceeded k. When it exceeds k, the sub-function must be again decomposable in order to be realized with k inputs. It is possible to check this at every stage of augmentation of the partially-formed sub-function and stop the trial when the number of variables exceeds k and start with a new variable.

Branch to step 2 at the end of Step 5. The procedure can be terminated when the given function can be expressed as a function of a number of sub-functions which may again be functions of still smaller sub-functions and so on, in a manner such that to realize any sub-functions no more than k disjoint inputs are necessary.

Some examples will be worked out below.

Example 5.3.1 $F = x_1x_2x_5 + x_1x_2x_6 + x_2x_5x_6 + x_1x_3x_5 + x_1x_3x_6 + x_3x_5x_6 + x_2x_4 + x_3x_4$

We shall test F for 3-unate decomposability.

Step 1. The function is unate.

Step 2. Each term has at least one variable that occurs in another term.

Step 3. The list of connected variables contains all variables of the function.

Step 4. Choose the term x_2x_4. Let $x_2x_4 = x_2 \mid H$ where $H = x_4$. The term x_3x_4 contains H; so, combining, $x_2x_4 + x_3x_4 = (x_2 + x_3)\, x_4$. A check reveals that x_3 and x_2 are interchangeable in the function.

Step 5. There is no third term containing x_4. Let $U_1 = x_2 + x_3$. Then F can be expressed as

$$F = (x_2+x_3)x_1x_5 + (x_2+x_3)x_1x_6 + (x_2+x_3)x_5x_6 + (x_2+x_3)x_4$$
$$= U_1\,(x_1x_5 + x_1x_6 + x_5x_6 + x_4)$$

Step 2. After separating the common factor U_1, consider the residual function $R = x_1x_5 + x_1x_6 + x_5x_6 + x_4$. The term x_4 occurs singly. Separating x_4, we have $U_2 = x_1x_5 + x_1x_6 + x_5x_6$.

Step 3. The function U_2 consists of 3 variables. Therefore F can be decomposed as $F = U_1(U_2 + x_4)$ where $U_1 = x_2 + x_3$ and $U_2 = x_1x_5 + x_1x_6 + x_5x_6$.

Example 5.3.2 $F = \bar{x}_1\bar{x}_3x_4x_5 + \bar{x}_2\bar{x}_3x_4x_5 + x_1x_2\bar{x}_4x_5 + x_3\bar{x}_4x_5$. We shall test F for 2-decomposability.

Step 1. The function is not unate.

Step 2. $F = x_5(\bar{x}_1\bar{x}_3x_4 + \bar{x}_2\bar{x}_3x_4 + x_1x_2\bar{x}_4 + x_3\bar{x}_4)$

$= x_5R$

when R is the residual function.

Step 3. The list of connected variables includes all the variables of the residual function.

Step 4. Let $\bar{x}_1\bar{x}_3x_4 = x_4 \mid \bar{x}_1\bar{x}_3 = x_4 \mid H$

Picking the term $\bar{x}_2\bar{x}_3x_4$ as it contains the literal $\bar{x}_3$ and combining, $\bar{x}_1\bar{x}_3x_4 + \bar{x}_2\bar{x}_3x_4 = \bar{x}_3(\bar{x}_2x_4 + \bar{x}_1x_4)$. The condition for grouping $\bar{x}_2x_4$ with $\bar{x}_1x_4$ in the same sub-function is satisfied.

Step 5. There is no other term with $\overline{x}_3$.

Let $\overline{x}_3(\overline{x}_2 x_4 + \overline{x}_1 x_4) = \overline{x}_3 x_4 \ (\overline{x}_2 + \overline{x}_1)$

$= \overline{x}_3 x_4 \ U_1$ where $U_1 = \overline{x}_2 + \overline{x}_1$

Then $R = \overline{x}_3 x_4 (\overline{x}_1 + \overline{x}_2) + \overline{x}_4 \ (x_1 x_2 + x_3)$

$= \overline{x}_3 x_4 U_1 + \overline{x}_4 (\overline{U}_1 + x_3)$

$= \overline{x}_3 x_4 U_1 + \overline{x}_4 \overline{U}_1 + \overline{x}_4 x_3$

Step 2. No modification.

Step 3. No modification.

Step 4. Let $\overline{x}_3 x_4 U_1 = \overline{x}_3 H$ where $H = x_4 U_1$

No other term of R contains any literal of H.

Reject this term and take $x_3 \overline{x}_4$ instead. Let $x_3 x_4 = x_3 G$. Combining $\overline{x}_4 \overline{U}_1$, we have, $x_3 \overline{x}_4 + \overline{x}_4 \overline{U}_1 = \overline{x}_4 (x_3 + \overline{U}_1)$. It is found that $\overline{U}_1$ can be included with x_3.

Step 5. There is no other term containing $\overline{x}_4$. Let $U_2 = x_3 + \overline{U}_1$.

Then $R = \overline{x}_3 x_4 U_1 + \overline{x}_4 (\overline{U}_1 + x_3)$

$= \overline{U}_2 x_4 + \overline{x}_4 U_2$

$= U_2 \oplus x_4$.

Thus F is 2-decomposable as

$F = x_5 R$ with

$R = U_2 \oplus x_4$, $U_2 = x_3 + \overline{U}_1$ and

$U_1 = \overline{x}_1 + \overline{x}_2$.

5.4 Synthesis of Non-disjunctive Network

Any arbitrary logic function can be realized by using an adequate number of k-input ULM's for any $k \geq 2$, with a suitable non-disjoint network structure. Therefore it is important to develop algorithm which attempts to reduce the cost of realization by minimizing the number of required logic elements. By a modification of the procedure suggested for disjunctive networks in section 5.3 it is possible to suggest a procedure for realizing a function with a non-disjunctive network. An advantage in following this method of synthesis is that the partially disjunctive feature of a function can be utilized to reduce the number of logical elements in the realization of the function. The modified algorithm is given below with the differences only mentioned.

Step 1. Same as in Section 5.3.

Step 2. Same as in Section 5.3.

Step 3. Same as in Section 5.3.

Step 4. The augmentation of a sub-function by including a new term is carried out according to the conditions mentioned in Section 5.3. When no second term is to be found for inclusion, the following rule is adopted.

Let P be a product of literals present in some term of the given function. Let Q be another such product of literals. For some terms $\{ PM_i \mid i = 1,2,..,M_i = \text{a product of literals} \}$ in F, in which P is present, let there be corresponding terms $\{ QM_i \mid i = 1,2,... \}$ in which Q is present. Let there be other terms of F involving P and Q in which the above is not true. In such a case, define a sub-function $U_j = P + Q$ and move to Step 5. It may be mentioned that in non-disjunctive case, it is possible to have some term of F containing as a partial product an implicant (not PI) of U_j.

Step 5. The sub-steps and conditions as given in section 5.3 are applied. After a sub-function U_j has been formed, the occurrences of the variables of U_j in F are replaced with U_j or $\overline{U}_j$ wherever possible.

The next step is to go back to Step 2 and repeat the procedure until the function can be decomposed into sub-functions of desirable size. An example is worked out below.

Example 5.4.1 $F = x_1x_2 + x_3x_4x_8 + x_3x_4x_6 + x_5x_6x_8 + x_6x_7x_8$

We shall realize F with 3-input or smaller ULM's.

Step 1. The function is unate.

Step 2. Separate x_1x_2 and get the residual function (R).

Step 3. The list of connected variables involves all variables of the residual function.

Step 4. Let $x_3x_4x_8 = x_3H$ where $H = x_4x_8$.

Combining $x_3x_4x_6$, we have

$$x_3x_4x_8 + x_3x_4x_6 = x_4 [x_3x_8 + x_3x_6]$$

Step 5. There is no other term with x_4.

Now, let $x_4(x_3x_8 + x_3x_6) = x_4x_3 (x_8 + x_6) = x_4x_3U_1$

where $U_1 = x_8 + x_6$. Then we have,

$$R = x_4x_3U_1 + x_5x_6x_8 + x_6x_7x_8.$$

Though U_1 involves the variables $\{ x_6, x_8 \}$, the occurrence of x_6x_8 in some terms cannot be replaced by U_1 or $\overline{U}_1$.

Step 2. The term $x_4x_3U_1$ is separated from the rest of the function.

Step 3. No modification.

Step 4. Let $x_5x_6x_8 = x_5 \mid x_6x_8 = x_5H$

Then $x_5x_6x_8 + x_6x_7x_8 = x_6x_8(x_5 + x_7)$

Step 5. No other term with x_6 or x_8 is present in the residual function. So, let $x_6x_8(x_5 + x_7) = x_6x_8U_2$

where $U_2 = x_5 + x_7$

Then $F = x_4x_3U_1 + x_6x_8U_2 + x_1x_2$

Step 2. Let $U_3 = x_4x_3U_1$, $U_4 = x_6x_8U_2$ and

$U_5 = x_1x_2$

Then $F = U_3 + U_4 + U_5$

The realization is shown in Figure 5.4.1.

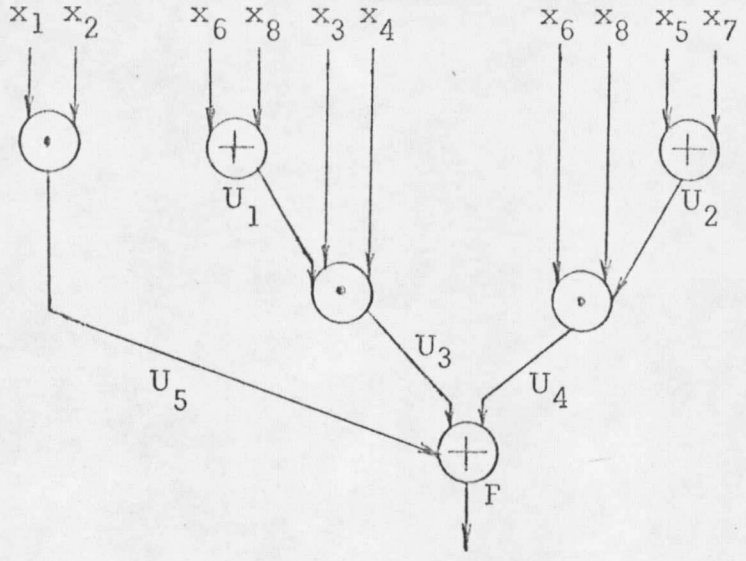


Figure 5.4.1 A Network to Realize $F = x_1x_2 + x_3x_4x_8 + x_3x_4x_6 + x_5x_6x_8 + x_6x_7x_8$

The above procedure of synthesis of non-disjunctive networks is given here as an extension of the test-synthesis method for disjunctive networks given before. A technique for finding the minimum number of k-input ULM's to realize a given function may require examination of a large number of possibilities. Roth and Karp[15] have dealt with similar problems under certain constraints.

Chapter 6

CONCLUSIONS

6.1 Summary

A study of the cellular bulk transfer system from the viewpoint of its logical capabilities has been presented in this thesis. The model adopted for the B.T. system consists of an input array, a mapping device, an output array and an output logic. The influence of various factors like flexibility of the mapping device, flexibility in the output logic, parallelism of operation has been investigated.

It has been shown in Chapter 2 that the system can be made logically universal with a proper combination of output logic and maps. Further, in realizing arbitrary logic, there exists a trade-off among the number of mapping operations, number of independent maps and amount of flexibility in the output logic such that increasing one of these quantities tends to decrease the others. A simple design for a B.T. system has been given for illustration.

In Chapter 3, the effect of introducing flexibility in the output logic by the use of flexible-logic cascades has been considered. Certain bounds on the number of required mapping operations (transposition type) have been obtained. A modification of the bulk transfer system is also discussed in which the input and output arrays consist of the primary or mapped inputs together with a set of functions derived from

the inputs by some built-in logic.

Chapter 4 deals with the topic of bulk transfer in parallel. Assuming the input array to be divided into many sub-arrays, an investigation has been made about the type of functions that can be realized by a number of B.T. units operating in parallel. Some properties of realizable functions have been proved and an algorithm has been presented for testing realizability of functions on the assumption that the input domain of each of the bulk transfer units working in parallel is fixed. In Chapter 5, the case of flexible input domain has been considered and an algorithm for the test-synthesis of logic functions on this basis has been developed.

Apart from a better understanding of the capabilities of a B.T. system, the above research has thrown light on the following topics.

i) Transformation of an arbitrary given function into Maitra-realizable functions.

The result given in section 3.2 shows that there is an exponential dependence of the maximum number of transpositions necessary to convert a given function into Maitra-realizable function on the number of variables. The same dependence is known to exist for the maximum number of cells necessary in a cutpoint or similar-type arrays to realize

an arbitrary function.

ii) Test-synthesis of logic functions realizable in disjunctive and simple non-disjunctive multi-level network.

The applicability of the synthesis algorithm for the disjunctive and non-disjunctive decomposition is not restricted to the particular B.T. system. They can also be used for the synthesis of any arbitrary digital network in multi-level form.

6.2 Scope for Further Research

There is scope of work on both theoretical and practical aspects of many problems in this area. Certain specific possibilities and questions which arose in connection with this research will be discussed.

It has been mentioned that each of the input and output arrays may consist of several cascades. In such systems, the input array may possibly represent the input domain of two-dimensional problems like the grid structure of the boundary-value problems or the input retina of a picture processing device or the matrices in ordinary computations. Useful primitive functions connected with the operations in solving these problems may be defined and implementation of these functions in the

B.T. system may be attempted. In particular, it will be of interest to find out an optimum number in order to do such problems. This problem has a partial similarity with the problem of minimization over Boolean graphs[15] in that while the number of logic modules at any stage of the mapping operation remain fixed, it is the number of stages in mapping (levels in Boolean graph) which must be minimized.

An interesting theoretical question can be raised with respect to the structure of Maitra-realizable functions. In section 3.4 the logical universality of a B.T. system consisting of a pair of cascades was proved on the assumption that each of these cascades contained at least one direct cell input more than the number of original variables $\{ x_1, \ldots x_n \}$. If each of the cascades consists of exactly (n-1) cells with n direct inputs, can we realize arbitrary function of n variables with the type of logic-free maps that can only permute the variables (independent or derived)? By the argument given in section 3.4, the system clearly can realize arbitrary function of (n-1) variables. Also, it can be easily seen that all functions belonging to the same equivalence class as any Maitra-realizable function of n-variables can be realized. Whether this system is logically universal or not is still an open question.

Some improvement in the algorithm presented in Chapter 5 may be attempted. This algorithm for grouping variables involves the procedure of taking one variable at a time and determining which subfunction includes it. For any variable, say x_j, which may be input to the last level ULM, the procedure shows that the required subfunction is the same as the original function after an exhaustive comparison has been made. Some faster way of determining the variables which are input to the lower-level ULM's may save this labour.

The technical aspect of bulk transfer of data as envisioned in this thesis should form an important part of further study and work in this area so that a practical bulk transfer system can be realized.

APPENDIX

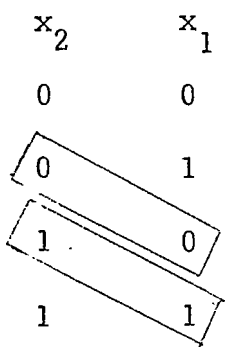
Appendix A

The next state equations for the variables to generate the permutation cycle $(m_1, m_2, \ldots m_{2n})$ are:

$$x_1' = \bar{x}_1 \ , \quad x_2' = x_2 \circledcirc x_1' \ ; \quad x_n' = x_n x_{n-1}' + \bar{x}_{n-1}' \quad (x_{n-1} \oplus x_n)$$

These are derived for the minterm sequence of which Table 2.3 is a replica for the case of three variables. Note that in any m_i such that $2 \leqq i \leqq 2^n$, the value of x_1 is the complement of that in m_{i-1}. The value of x_1 in m_1 is the complement of that in m_{2n}. Therefore, for the cycle $(m_1, m_2, \ldots m_{2n})$ we may write $x_1' = \bar{x}_1$.

For x_2', we see that x_2' can be expressed in terms of the present value of x_2, present value of x_1 and the next value x_1' of x_1. Note the minterms having $x_2 = 1$.



From inspection of the above diagram, we have, $x_2' = 1$ if $x_2 = 0$ and $x_1' = 0$ or if $x_2 = 1$ and $x_1' = 1$.

Hence $x_2' = x_2 \circledcirc x_1'$

Similarly for x_3', we inspect the terms having $x_3' = 1$

x_3	x_2	x_1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Then we obtain the following truth table (combinations for which x'_3 is true are only shown).

x_3	x_2	x_2'	x_3'
0	1	0	1
1	0	0	1
1	0	1	1
1	1	1	1

From this table, we have

$$x_3' = x_3 x_2' + \overline{x_2'}\ (x_3 \oplus x_2)$$

It is noticed that of the eight minterms of the three variables $\{x_3, x_2, x_2'\}$, four give a true value for x_3'. This property holds true for higher

variables like $x'_4, x'_5, \ldots x'_n$. Therefore, for $n \geq 3$, x'_n can be expressed as

$$x'_n = x_n x'_{n-1} + \overline{x'_{n-1}} (x_n \oplus x_{n-1})$$

To get the equations for generating Φ, note that the map is similar to the cyclic permutation except that the last minterm $x_1 x_2 \ldots x_n$ must be mapped into itself. An AND gate is used to detect the occurrence of this minterm and the output of the gate is used in that event to obtain the proper next states of all variables.

Therefore for generating Φ, we have

$$x'_1 = \overline{x}_1 + x_1 x_2 \ldots x_n; \quad x'_2 = (x'_1 \ominus x_2) + x_1 x_2 \ldots x_n$$

For $n \geq 3$, $x'_n = \overline{x}'_{n-1} (x_{n-1} \oplus x_n) + x'_{n-1} x_n + x_1 x_2 \ldots x_n$

Appendix B

The proofs of the generalized theorems involve the same kind of argument as in the special cases. These are sketched below.

Proof of Theorem 4.2.1': The first step is to prove that either $P_i \subseteq U_i$ or $P_i \subseteq \overline{U}_i$. Assuming this to be false, let there be minterms $\{M_1, M_2\}$ of the variables of U_i such that $M_1, M_2 \subseteq P_i$ and $M_1 \subseteq U_i$ and $M_2 \subseteq \overline{U}_i$. Expressing $P_a P_b \ldots P_i \ldots P_k = HP_i$ where H represents the products of literals other than those contained in P_i, it can be shown using the same arguments as in the proof of 4.2.1, that H is an implicant of F. It is a contradiction to the assumption that HP_i is a PI of F. Therefore $P_i \subseteq U_i$ or $P_i \subseteq \overline{U}_i$. Next, assume WLOG, $P_i \subseteq U_i$ and P_i is not a PI of U_i. Let $P_i \subset P_i^* \subseteq U_i$. This again leads to the contradictory result that HP_i^* is an implicant of F. So P_i must be a PI of U_i. Q.E.D.

Proof of Theorem 4.2.2': With the given condition, assume WLOG, $P_i \subseteq U_i$ but $Q_i \subseteq \overline{U}_i$. It leads to the conclusion that X is an implicant of F which is contradictory. Therefore with the conditions given, $Q_i \subseteq U_i$ if $P_i \subseteq U_i$. The other part follows from Theorem 4.2.1'.

Converse Case: WLOG let $P_i, Q_i \subseteq U_i$. When $P_i X$ is true, F = 1. Therefore F must be true when $Q_i X$ is true, because F can be expressed

as a function of U_i and X. Thus $Q_iX \subseteq F$. For the second part, assume Q_iX is not a PI of F and $Q_iX \subset Q_i^*X^* \subseteq F$. If $Q_i \subset Q_i^*$, then we are led to the contradictory conclusion that Q_i is not a prime implicant of U_i. If $X \subset X^*$, then it leads to the conclusion that P_iX is not a PI of F. Therefore $Q_i = Q_i^*$, $X = X^*$ and Q_iX is a PI of F. Q.E.D.

Proof of Corollary 1': If F is a complete sum of PI's, then the proof follows from Theorem 4.2.2'. Let F be an incomplete set of PI's. WLOG, let M be a true minterm of U_i which is not contained in the PI's of U_i obtained under A_i. Let P_iX be a PI of F such that $P_i \subseteq U_i$. Then $MX \subseteq F$. Therefore, MX must be included in some PI's of F. Since X is not a PI, there exists Q_i under A_i such that $M \subseteq Q_i$. By Theorem 4.2.1' Q_i is a PI of U_i. Therefore, the assumption $M \subseteq U_i$ but not included in any PI of U_i under A_i is false. Q.E.D.

Proof of Theorem 4.2.3': Assume that $P_iP_j \ldots P_k$ and $Q_iQ_j \ldots Q_k$ are two product terms of F. Applying the two conditions, mentioned in the proof of Theorem 4.2.3, to the set of product terms of F to check if they contain any implicants which are not prime implicants, it can be shown that condition (i) cannot be true for the pair $P_iP_j \ldots P_k$ and $Q_iQ_j \ldots Q_k$, because it would lead to the conclusion that certain P_j and Q_j satisfied the condition. This would violate the assumption that P_j and Q_j are prime implicants of a constituent function U_j. Condition

(ii) also cannot be true because it would imply that certain $P_j \subset Q_j$, or $Q_j \subset P_j$ which would again violate the said assumption. Q.E.D.

Proof of Theorem 4.2.4': Let U be unate in U_i. Assume that the sum of the terms under A_i equals 1. Then either U_i is trivial or PI's of both U_i and $\overline{U}_i$ are present under A_i. The first possibility is against the assumption that F is non-degenerate. The second possibility implies that U is not unate in U_i. Contradiction. Therefore if U is unate in U_i, $\sum_j P_i^j \neq 1$.

Converse Case: Let $\sum_j P_i^j \neq 1$ and U is not unate in U_i. Then PI's of both U_i and $\overline{U}_i$ must occur in the PI's of F. By Corollary 1', the PI's of U_i and $\overline{U}_i$ occurring under A_i are sufficient to define U_i and $\overline{U}_i$. Therefore the sum of these PI's = $U_i + \overline{U}_i = 1$. This is a contradiction. Hence the theorem. Q.E.D.

Proof of Theorem 4.2.5': Assume P_iX, $Q_i \subseteq F$. Let P_i, Q_i be PI's of $U_i(\overline{U}_i)$. Then by Theorem 4.2.2', Q_iX must be a PI of F. Contradiction. Let $P_i \subseteq U_i$ and $Q_i \subseteq \overline{U}_i$. Then F can be expressed in terms of U_i as $F = U_iX + \overline{U}_i + R$ where R is some residual function. Taking consensus between U_iX and $\overline{U}_i$, we obtain X as an implicant of F. Contradiction. Hence the theorem. Q.E.D.

Proof of Theorem 4.3.1: To prove first that $P_1X_1 \subseteq U_1$ $(\overline{U}_1)$ and $X_1P_2 \subseteq U_2$ $(\overline{U}_2)$. Consider the possible cases as in the proof of Theorem 4.2.1. It can be seen in an exactly similar manner that if $P_1X_1 \not\subseteq U_1$ and $P_1X_1 \not\subseteq \overline{U}_1$ or $X_1P_2 \not\subseteq U_2$ and $X_1P_2 \not\subseteq \overline{U}_2$, then $P_1X_1P_2$ cannot be a PI of F. Next, to show P_1X_1 as a PI of $U_1(\overline{U}_1)$, assume $P_1X_1 \subset P_1^*Y_1^*$ where P_1^* and Y_1^* are subsets of the literals of P_1 and X_1 respectively. It follows that $P_1X_1P_2$ is not a PI of F. Contradiction. Similarly it can be shown that if X_1P_2 is not a PI of $U_2(\overline{U}_2)$ it leads to a contradiction. Hence the theorem. Q.E.D.

Proof of Theorem 4.3.2: WLOG, let $P_1X_1 \subseteq U_1$. Now if $Q_1Y_1 \subseteq \overline{U}_1$, it leads to the conclusion that X_1P_2 is an implicant of F, which is contradictory. The other part follows from Theorem 4.3.1.

Converse part: WLOG, let P_1X_1, $Q_1X_1 \subseteq U_1$ and $P_1X_1P_2 \subseteq F$. Assume, WLOG, $X_1P_2 \subseteq U_2$. In a manner similar to Theorem 4.2.2, it can be shown that when $Q_1X_1P_2$ is true, $F = 1$. Thus $Q_1X_1P_2 \subseteq F$. To prove the other part note that since $P_1X_1P_2$ is a PI of F and P_1X_1 a PI of U_1, hence X_1P_2 is a PI of U_2 (by Theorem 4.3.1). Now if $Q_1X_1P_2$ is not a PI of F and there exists certain $Q_1^*X_1^*P_2^* \supset Q_1X_1P_2$, then, it leads to the contradiction that either Q_1X_1 is not a PI of U_1 or X_1P_2 is not a PI of U_2. Hence the result. Q.E.D.

Generalization of Theorems 4.3.1 and 4.3.2

Let P_i, Q_i, R_i etc--letters with subscript i represent products of the literals of the variables of D_i for $i = 1,2,\ldots s$; α_i, β_i, Y_i etc. for the variables of C_i, for $i = 1,2,\ldots(s-1)$. Let U_i be a function on the variables of A_i and U be the function on U_i's for $i = 1,2,\ldots,s$.

Theorem 4.3.1': Let $P_i\alpha_i P_{i+1}\alpha_{i+1}\ldots P_j\alpha_j P_{j+1}\ldots P_{k+1}$ be a PI of F. Then $P_j\alpha_j$ is a PI of $U_j(\overline{U}_j)$ and $\alpha_j P_{j+1}$ is a PI of $U_{j+1}(\overline{U}_{j+1})$ for $j = i,\ldots k$.

Theorem 4.3.2': If P_iA and Q_iA be two PI's of F where A, B represent products of literals not containing any literals of D_i, then $Q_i\alpha_i \subseteq U_i(\overline{U}_i)$ if $P_i\alpha_i \subseteq U_i(\overline{U}_i)$ where α_i is a subset of the literals of A. Also, $Q_i\alpha_i$ is a PI of $U_i(\overline{U}_i)$ if $P_i\alpha_i$ is a PI of $U_i(\overline{U}_i)$. Conversely if $P_i\alpha_i$, $Q_i\alpha_i \subseteq U_i(\overline{U}_i)$, then $Q_iA \subseteq F$ if $P_iA \subseteq F$ where A contains the literals of α_i. Further if $P_i\alpha_i$ and $Q_i\alpha_i$ are PI's of U_i and P_iA is a PI of F, then Q_iA is a PI of F.

The proofs of these theorems may be obtained by arguments similar to those in Theorems 4.3.1 and 4.3.2.

LITERATURE CITED

1. Anderson, L.K., "Holographic Optical Memory for Bulk Data Storage" Bell Laboratories Record, November 1968. pp 319-325.

2. Ashenhurst, R.L., "The Decomposition of Switching Functions," Proc. Int. Symposium on the Theory of Switching Annals of the Computation Laboratory of the Harvard University,Vol-29,pp 74-116.

3. Birkhoff, G., and Maclaine, S., A Survey of Modern Algebra, New York, Macmillan & Co., 1965.

4. Curtis, H.A., The Design of Switching Circuits, D. Van Nostrand Co., Inc., Princeton, New Jersey, 1962.

5. Elspas, B., et. al., Properties of Cellular Arrays for Logic and Storage, Scientific Report 3, SRI Project 5876, July 1967 (AFCRL-67-0463).

6. Fraleigh, J.B., A First Course in Abstract Algebra, Addison-Wesley Publishing Co., 1967.

7. Hawkins, J.W., and Munsey, C.J., "A Two-dimensional Iterative Network Computing Technique and Mechanization," Proc. 1962 Workshop on Computer Organization, pp 93-125, 1963.

8. Holland, J.H., "Iterative Circuit Computers," Proc. Eastern Joint Computer Conference, pp. 108-113, 1957.

9. Maitra, K.K., "Cascaded Switching Networks of Two-input Flexible Cells," IRETEC, Vol. EC-11, No. 2, pp 136-143, April 1962.

10. McCluskey, E.J., Introduction to the Theory of Switching Circuits, McGraw Hill Book Co., pp 165-174, 1965.

11. Minnick, R.C., "A Survey of Microcellular Research," J.A.C.M., Vol. 14, pp 203-261, April 1967.

12. Minnick, R.C., "Cutpoint Cellular Logic," IEEE Trans. on Electronic Computers, Vol. EC-13, pp 685-698, December 1964.

13. Minsky, M. and Papert, S., Perceptions, The MIT Press, 1969.

14. Mukhopadhyay, A., "Unate Cellular Logic," IEEE Trans. on Computers Vol. C-18, pp 114-121, February 1969.

15. Roth, J.P., and Karp, R.M., "Minimization over a Boolean Graph," IBM Journal of Research and Development, Vol. 6, No. 2, pp 227-238, April, 1962.

16. Short, R.A., "Two-rail Cellular Cascades," Proc. of the AFIPS Fall Joint Computer Conference, Vol. 27, Part I, pp 355-369, 1965.

17. Unger, S.H., "A Computer Oriented Toward Spatial Problems," Proc. IRE, 10 (Oct. 1958), pp 1744-1750.

RESEARCH REPORTS AND PAPERS ON WHICH THE THESIS HAS BEEN BASED.

1. Minnick, R.C., Thurber, K.J., Mukhopadhyay, A., Roy, K.K., Cellular Bulk Transfer Systems, Final Report, MSU Project ERL-8-0009-601, Montana State University, Bozeman, Montana AFCRL 68-0497, October 1968.

2. Mukhopadhyay, A., Schmitz, G., Thurber, K.J., and Roy, K.K., Minimization of Cellular Arrays, Final Report, NSF Contract GJ-158, Electronics Research Laboratory, Endowment and Res. Foundation, Montana State University, Bozeman, Montana, September 1969.

3. Roy, K.K., "Decomposition of Logic Functions for Realization in Multi-Level ULM Networks"--to be presented at the Third Hawaii International Conference in System Sciences, January 1970.