Alternative approaches to combinational and sequential logic design.

Picton, P. D.

Award date:
1982

Awarding institution:
University of Bath

ALTERNATIVE APPROACHES TO COMBINATIONAL AND SEQUENTIAL LOGIC DESIGN

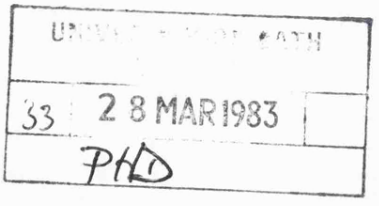
submitted by

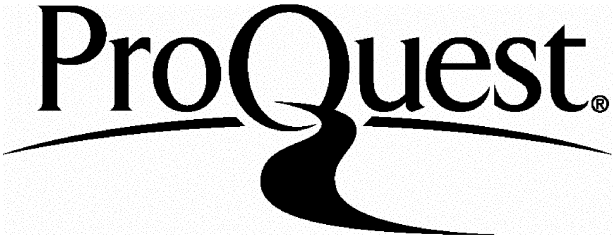
P.D. PICTON

for the degree of Ph.D.

of the University of Bath

1982





SUMMARY

This thesis is concerned with the developments firstly in combinational logic, where the problems involved in multi-threshold realisations are examined, and secondly in sequential logic, where the specific type of system known as serial input logic is considered.

Chapter 1 is an introduction which discusses the areas covered by this thesis and indicates their relationship to one another within the general framework of logic design.

Chapter 2 sets out to extend the number of functions that can be realised by a multi-threshold logic gate with near optimal weights and thresholds. Use is made of the already existing spectral translation method of obtaining a single threshold solution with additional exclusive-or gates, a multi-threshold solution being obtained by algebraic manipulation of the weights and threshold. The mathematical basis that enables this to be done is derived and examples given.

Chapter 3 discusses the possible advantages of using a multi-threshold logic gate within a charge-coupled device over the alternative Boolean AND/OR and quaternary logic gates. The fundamental operations of a charge-coupled device are reviewed, and the tolerancing problems that result from the charge transfer inefficiency and voltage fluctuations are considered as limiting factors on the logical complexity of the gate.

Chapter 4 is concerned with the subject of serial input logic. Initially it sets out to define serial input logic in terms of a general sequential system, and then goes on to show that with regard to state reduction using compatibles it is unique since it only requires the derivation of the implied maximal compatibles. Furthermore, a modular realisation is given, where the design procedure consists

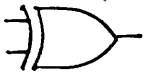
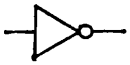
of the use of reverse response trees. Various labelling schemes are considered and finally one is considered that guarantees an optimal solution.

Throughout the thesis emphasis is placed on finding general solutions whenever possible, so that not only do they apply to the situations described herein, but also may prove useful in as yet undeveloped areas of research.

LIST OF SYMBOLS

List of Symbols

n	Number of input variables to a logical function
$x_i, i = 1$ to n	Binary input variable, $x_i \in \langle 0,1 \rangle$
$f(x), f(x_1, x_2, \ldots, x_n)$	Function of the x_i variables, $f(x) \in \langle 0,1 \rangle$
$\left[\; \right]$	Matrix
$\left[\right]$	Column matrix
$+$	Arithmetic addition or logical OR
$\cdot$	Arithmetic product or logical AND (omitted when no ambiguity arises)
$r_i, i = 0,1,\ldots,12..n$	Rademacher-Walsh spectral coefficient
$\Longleftrightarrow$	Interchange
iff	If and only if
-	Don't care
V	Voltage
Q	Charge
ϕ_s	Surface potential
ϕ_i	Clock phase
ϵ	Charge transfer inefficiency
β	Fractional voltage fluctuation
$\sim$, $(\not\sim)$	Compatibility, (incompatibility)
$\subseteq$	A subset of
$\subset$, $(\not\subset)$	A proper subset of, (not a subset of)
ϵ	Contained in
$p_i, i = 0$ to $2^n - 1$	The i^{th} minterm of a function $f(x)$, where the corresponding row of input variables in the truth table read as a binary number, x_n being the least significant digit

$\cup$	Union
J_i, K_i, R_i, S_i, T_i	Rademacher-Walsh spectra of the characteristic functions of J-K, R-S, and T bistables, respectively
	AND gate
	OR gate
	Exclusive-or gate
	Inverter
	Gate
	Drain } Used in C.C.D.'s
	Source

CONTENTS

CONTENTS

CHAPTER 1

INTRODUCTION

1. Introduction

Two of the observable current trends in logic design are:

a) to accept that physical miniaturisation of circuits is a finite process and hence the search for devices with greater logical power per unit area of chip,

b) to simplify and unify logic design procedures by the use of universal logic modules.

For the former, the bulk of research has been concentrated on multi-valued logic (usually ternary, 3-valued, or quaternary, 4-valued) and threshold logic. The subject of threshold logic has been extensively researched over the past two decades but its application has been limited, mainly due to the relatively few functions that can be directly realised with single threshold devices.

Chapter 2 of this thesis attempts to extend the use of threshold logic by the utilisation of multi-thresholds. To date the major features and operations of threshold logic theory are as follows [1]:

a) the characterisation of threshold functions with $n + 1$ coefficients called the Chow's parameters.

b) Lists of Chow's parameters are available for $n \leq 8$ which can be used to determine whether or not a function is threshold realisable, and if so gives the optimised weights and thresholds to do so.

c) the ability to invert a threshold function and/or any of its input variables by the negation of the relevant weights; also the ability to permutate the input variables.

d) the classification of all functions including the non-threshold ones by the use of the 2^n coefficients known as the Rademacher-Walsh spectrum, functions in the same class being related by the use of "spectral translation".

e) the division of these classes into threshold and non-threshold classes, the former consisting of functions which are, or upon the application of spectral translation become, threshold functions, the latter functions being unable to be translated into a threshold class.

f) lists of the Rademacher-Walsh spectra of all functions of $n \leq 4$ giving "optimised" weights and thresholds for a multi-threshold realisation are available.

The aim of Chapter 2 is to extend the number of manipulations that can be done to the weights and thresholds of a function in order to obtain dissimilar functions. These manipulations can then be used to convert functions contained in the threshold class into multi-threshold solutions, i.e. convert functions which are realisable with a single threshold gate plus any exclusive-or gates (to perform the equivalent in the Boolean domain as spectral translation in the spectral domain) into multi-threshold form. The reasons for wanting a multi-threshold solution as opposed to the single threshold with additional hardware are given, and also proposed is the conversion of at least some of the functions contained in the non-threshold classes into a multi-threshold solution.

Another area of development in threshold logic has been in the design of the gate itself. Many have been developed using various technologies, two of the most promising being the Digital-Summation-Threshold-Logic gate (D.S.T.L.) and the I^2L gate [1]. However, recently

a need has been expressed for the incorporation of logic in charge-coupled devices (C.C.D.'s), with a view to developing a system comprising both a digital memory and a logic processor on a single chip [2]. The previously mentioned threshold logic gates are incompatible with C.C.D. systems since the data would have to be taken out of the C.C.D. system, logic operations performed, and then the data returned.

The hitherto proposed alternative logical approaches which can be incorporated into C.C.D. technology are either arrays of binary AND/OR gates [2] or quaternary logic gates [3]. Chapter 3 sets out to indicate that the operations used in these logic gates, such as charge input, transfer, and overflow, can be applied equally well to a C.C.D. threshold logic gate. However, an additional feature is required for a threshold logic gate, namely the use of multi-levels of charge when the number of levels is greater than four, quaternary logic having already illustrated the use of up to four levels. A justification of this is therefore also included in Chapter 3.

A multi-threshold logic gate structure is proposed and details of its mode of operation and timing diagrams are included. Its drawbacks are discussed, such as tolerancing problems and voltage rail fluctuations. The gate is then compared not only with the alternative C.C.D. logic gates but also with the previously suggested threshold logic gates.

An interesting feature of the gate is that a clock is present and therefore it represents a synchronous system. The application of threshold logic to synchronous systems has received some attention but it is not pursued in this thesis. Of note, however, is the fact that the gate can also be described as having clocked parallel inputs. An alternative approach would therefore be to use clocked serial inputs.

Such a system with serial inputs would have to be synchronous since the data would be arriving on the same input line, and therefore each piece of data would have to occupy its own time slot. Thus the data can be envisaged as a word-formatted n bit serial stream. To date such systems have received little attention, that which it has received being confined to mode-controlled logic [4]. This area is therefore examined in more depth in Chapter 4.

The intention of the first part of Chapter 4 is to define the subject more broadly and to examine it from the point of view of a general sequential system. In particular, the field of state reduction using compatibles is studied. The chapter sets out to prove that a serial input logic system is a special class of sequential system where the number of states can be reduced by merely considering the maximal compatibles, as opposed to the usual situation where a great deal of complex and time consuming manipulation is required.

The chapter goes on to consider the use of modules in the design of serial input systems. As stated at the beginning of this introductory chapter, the use of universal logic modules is an important aspect of logic design in both combinational and sequential systems, the difference being that in sequential logic the modules usually contain delay elements whereas in combinational they do not. Of particular interest is the work that has been done on the application of modular networks to the design of any sequential system, where the network needs at most one feedback loop [5]. Furthermore, it has been shown that some machines, known as definite machines, do not need any feedback and are therefore more easily realisable and testable. The chapter shows that serial input logic systems are definite machines and therefore they too do not require any feedback.

More specialised design procedures are considered where the need for flow diagrams, state tables, state reduction, and state assignment are dispensed with. These consist of reverse response trees, which are particularly easy to use, and it is shown that they can be drawn directly from either the Boolean expression of the required function or from its truth table vector.

The chapter then considers some additional features and shows that by relaxing some of the initial conditions some of the more commonly encountered systems such as code converters and serial adders can be designed.

Overall, the thesis covers topics which involve many different areas of combinational and sequential logic. Figure 1.1 depicts schematically the relationship between these topics and indicates those that appear in the main chapter developments of this thesis.

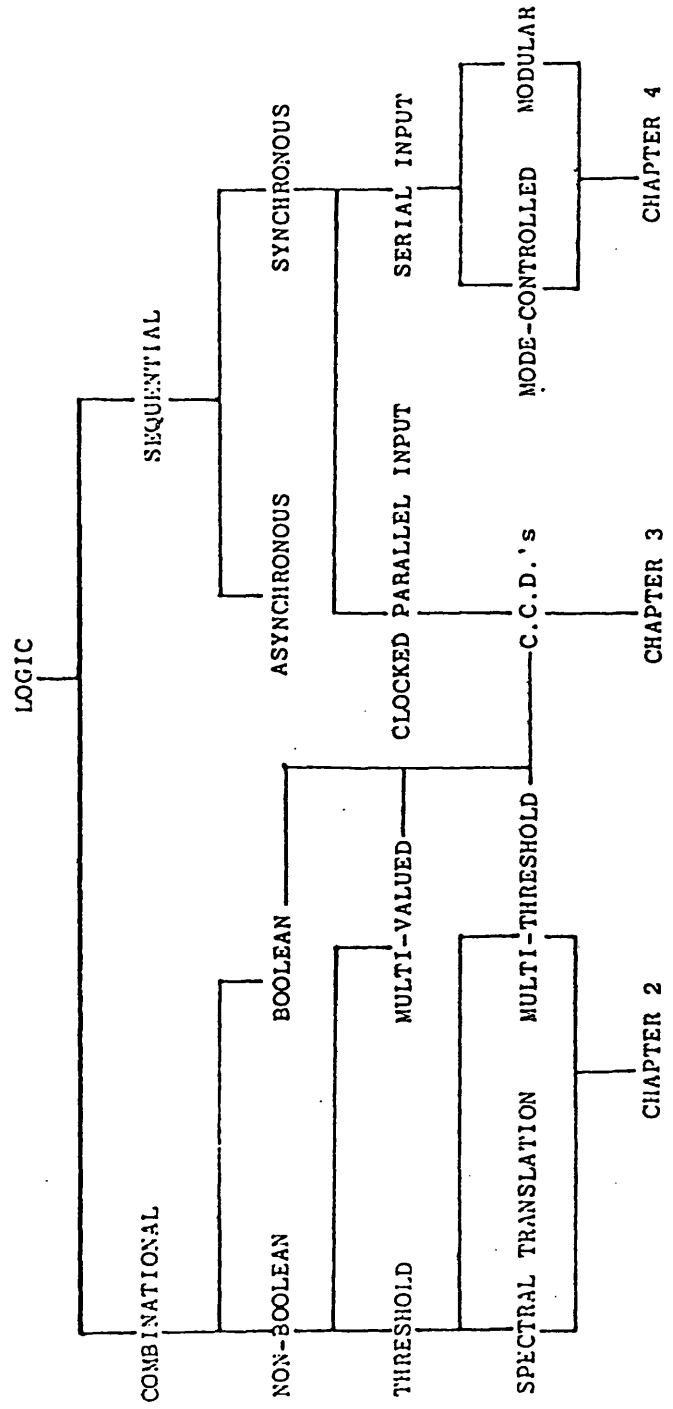


Figure 1.1 Relationship between topics discussed in each chapter

References

1. Hurst, S.L.: "The Logical Processing of Digital Signals", (Crane-Russak, New York, and Edward Arnold, London, 1978).

2. Zimmerman, T.A., Allen, R.A., Jacobs, R.W.: "Digital charge-coupled logic (D.C.C.L.)", IEEE J. Solid State Circuits, SC-12, 5, pp.473-485, Oct. 1977.

3. Kirkhoff, H.G., Kijstra, H.: "The application of C.C.D.'s in multiple-valued logic", Proc. 5th International Conference on Charge Coupled Devices, pp.304-309, 1979.

4. Daws, D.C., Jones, E.V.: "Mode-controlled serial logic systems", IEEE International Symposium on Circuits and Systems, (Chicago), pp.902-905, 27th-29th April 1981.

5. Friedman, A.D., Menon, P.R.: "Theory and Design of Switching Circuits", (Computer Science Press-Digital Design Series, 1975).

CHAPTER 2

THRESHOLD LOGIC

2. Threshold Logic

2.1 Single Threshold Logic

Threshold logic has received much attention over the past twenty years [1,2,3,4] but has enjoyed very limited application. There are a number of reasons for this which will presently be discussed, but first a definition of threshold logic.

A threshold logic gate is a non-vertex gate which receives binary inputs and yields a binary output and which obeys the following equation:

$$f(x) = 1 \text{ iff } \sum_{i=1}^{n} a_i x_i \geq t$$
$$= 0 \text{ otherwise} \qquad 2.1$$

where a_i is an integer weight associated with the input variable x_i and t is an integer threshold value.

Figure 2.1 shows the conventional symbol for a threshold logic gate.

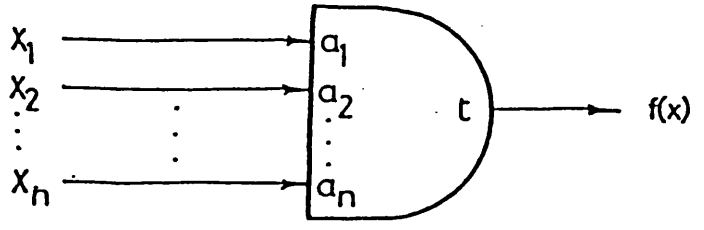


Figure 2.1 Symbol for a threshold logic gate

The first limitation of threshold logic is that although equation 2.1 is a very simple equation, it is extremely difficult to apply to a general Boolean function, firstly because not all are threshold functions, secondly because it is difficult to determine those that are from those that are not, and thirdly because of the problems of

finding suitable weights and thresholds for those that are. To a large extent these problems have been overcome by the use of tables of Chow's parameters [5,6]. Essentially, Chow showed that if a function is a threshold function then it can be uniquely characterised by n + 1 coefficients. For a given function if these coefficients are calculated, their moduli taken, and then set in descending order, they can be compared to available lists of known parameters of threshold functions, and thus determine whether or not the function is a threshold one. The tables also include optimised weights and thresholds which have been derived empirically. However, these tables are restricted to up to eight input variables only since the size of the tables become unmanageable for a greater number [7,8]. Appendix A gives a listing of Chow's parameters for up to six variables.

The second limitation of threshold logic is the physical realisation of the gate itself. Many gates have been suggested, more of which will be mentioned in the next chapter where one is proposed which consists of a charge-coupled device. However, none of these have ever been accepted commercially which is not due to any fault of the gates themselves, but to industries' lack of enthusiasm to support something having such a limited application, namely the realisation of the restricted class of simple threshold functions of up to eight input variables. Therefore work is still required to be done to extend the number of functions that it can realise. Two ways of doing this are by the use of either spectral translation or multi-threshold logic, both of which will be discussed in detail in the following sections.

2.2 Spectral Translation

A spectrum is an alternative description of a function to the more conventional truth table which is used to highlight particular features that were previously concealed [1,4,9]. It is obtained by the following equation:

$$\left[T\right].F\right] = R\right] \qquad 2.2$$

where $R\right]$ is the column matrix of the spectral coefficients

$F\right]$ is the truth table vector of the function recoded in $\langle 1,-1\rangle$ instead of $\langle 0,1\rangle$

and $\left[T\right]$ is the 2^n by 2^n transform matrix.

Although there are many different transform matrices [10] one of the most commonly used is the Rademacher-Walsh, which will be the only once considered from here on. As an example of this matrix Figure 2.2 shows the case for n = 3.

Equivalent to the row entries	Transform matrix									Spectral function coefficients
x_0	1	1	1	1	1	1	1	1		r_0
x_1	1	1	1	1	-1	-1	-1	-1		r_1
x_2	1	1	-1	-1	1	1	-1	-1		r_2
x_3	1	-1	1	-1	1	-1	1	-1		r_3
$x_1 \oplus x_2$	1	1	-1	-1	-1	-1	1	1	.F =	r_{12}
$x_1 \oplus x_3$	1	-1	1	-1	-1	1	-1	1		r_{13}
$x_2 \oplus x_3$	1	-1	-1	1	1	-1	-1	1		r_{23}
$x_1 \oplus x_2 \oplus x_3$	1	-1	-1	1	-1	1	1	-1		r_{123}

Figure 2.2 Rademacher-Walsh transform

It can be seen from Figure 2.2 that each row of the transform matrix, when converted back to the original $\langle 0,1 \rangle$ coding, is equivalent to an input variable or the exclusive-or combination of input variables, which indicates the convention of the subscript labelling of the spectral coefficients.

If a function is a threshold function then the first n + 1 coefficients, namely $r_0 r_1 \ldots r_n$ are equivalent to the Chow's parameters discussed earlier, and therefore uniquely describe that function. However, functions which are not threshold require up to the full 2^n coefficients of the spectrum to describe them.

The inverse transform is readily available so that functions are retrievable from their spectra as follows:

$$2^{-n}.\left[T\right]^{-1}.R\right] = F\right] \qquad 2.3$$

where $\left[T\right]^{-1}$ is obtained from the relationship:

$$2^{-n}.\left[T\right]^{-1}.\left[T\right] = \left[I\right] \text{ the unit matrix} \qquad 2.4$$

As an example of a spectrum and its uses consider the following logic function:

$$f(x) = x_1.\bar{x}_2 + \bar{x}_1.x_2 + x_2.x_3$$

Applying the transform of equation 2.2:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 2 \\ 2 \\ 6 \\ -2 \\ -2 \\ 2 \end{bmatrix}$$

Whence:

r_0	r_1	r_2	r_3	r_{12}	r_{13}	r_{23}	r_{123}
-2	2	2	2	6	-2	-2	2

If the first n + 1 coefficients, in this case $r_0 r_1 r_2 r_3$, are taken, the modulus of each found, and then placed in descending order, they can be compared with the table of Chow's b parameters in the list for $n \leq 3$ (see Appendix A). It is found that there is no entry of 2222, and so the function is non-threshold. However, the spectrum contains the coefficient $r_{12} = 6$, so that if in some way this could be included in the set of n + 1 coefficients then the ordered set of their moduli would be 6222 which is among the entries in the table. One way of doing this is by "moving" or "translating" the coefficients. This is therefore known as spectral translation [4,11,12], which may be divided into two distinct operations namely output and input spectral translation.

2.2.1 Output Spectral Translation

This is achieved by taking the exclusive-or of the function f'(x) with an input variable to obtain a new function f(x) which is related by equation 2.5.

$$f(x) = f'(x) \oplus x_i \qquad 2.5$$

The effect of this operation on the spectral coefficients of f'(x) is to delete i from the subscripts of coefficients which already have i in them, and to append i if they have not. In other words, pairs of coefficient values are exchanged according to the following rule:

$$\begin{aligned} r_0 &\Longleftrightarrow r_i \\ r_j &\Longleftrightarrow r_{ij} \\ r_{jk} &\Longleftrightarrow r_{ijk} \\ &\text{etc.} \end{aligned} \qquad 2.6$$

Clearly this can be extended to include more than one input variable, in which case the subscript i in equation 2.6 is replaced by the combined subscripts of the variables involved.

In the previous example, if the exclusive-or is taken of the function and the input variable x_1, then the new spectrum is:

r_0	r_1	r_2	r_3	r_{12}	r_{13}	r_{23}	r_{123}
2	-2	6	-2	2	2	2	-2

Comparing with the list of Chow's parameters, the relevant entry is:

parameter subscripts	0	1	2	3
b parameters	6	2	2	2
a parameters	2	1	1	1

When the a parameters are reordered and the signs reintroduced they become:

a_0	a_1	a_2	a_3
1	-1	2	-1

The threshold is found using the following equation:

$$t = \frac{1}{2} \cdot \left(\sum_{i=0}^{n} a_i + 1 \right) = 1 \qquad 2.7$$

Figure 2.3 shows the final realisation of the function.

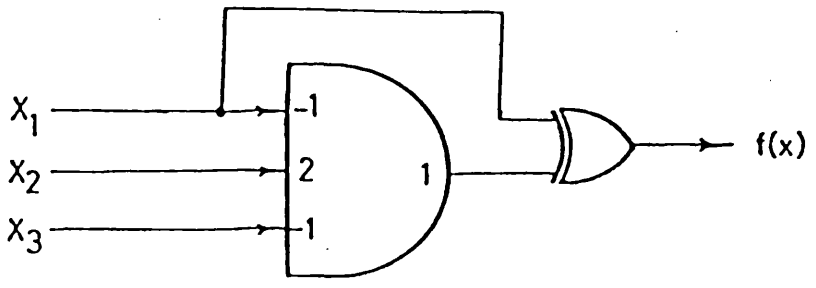


Figure 2.3 Realisation of the function $f(x) = x_1 . \overline{x}_2 + \overline{x}_1 . x_2 + x_2 . x_3$ using output spectral translation.

2.2.2 Input Spectral Translation

This converts a function f'(x) into a new function f(x) by replacing one of its input variables with the exclusive-or of the same input variable and one other as in equation 2.8.

$$f(x_1,x_2,\ldots,x_i,x_j,\ldots,x_n) = f'(x_1,x_2,\ldots,x_i \oplus x_j,x_j,\ldots,x_n) \qquad 2.8$$

The effect on the spectral coefficients of f'(x) is that if a coefficient contains i in its subscript then delete j if it is already present and append j if it is not. In other words, pairs of coefficient values are exchanged in accordance to the following rules:

$$r_i \Longleftrightarrow r_{ij}$$
$$r_{ik} \Longleftrightarrow r_{ijk} \qquad 2.9$$
$$\text{etc.}$$

This can be extended to the situation where an input variable is replaced by the exclusive-or of itself and more than one other input variable, in which case j in equation 2.9 is replaced by the combined subscripts of the other variables involved.

Again consider the previous example, only this time replace x_1 by $x_1 \oplus x_2$. The new spectrum is:

r_0	r_1	r_2	r_3	r_{12}	r_{13}	r_{23}	r_{123}
-2	6	2	2	2	2	-2	-2

The same Chow's b and a parameters are found as in the case of output spectral translation, but when reordered and the signs replaced the set of weights is found to be:

a_0	a_1	a_2	a_3
-1	2	1	1

and a threshold using equation 2.7 of:

$$t = 2$$

Figure 2.4 shows the final realisation of the function.

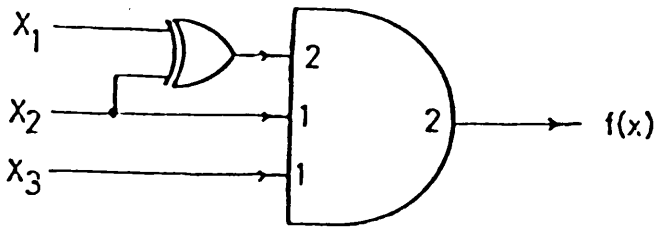


Figure 2.4 Realisation of the function $f(x) = x_1.\bar{x}_2 + \bar{x}_1.x_2 + x_2.x_3$ using input spectral translation.

2.2.3 Rademacher-Walsh Spectral Classification

A scheme for the classification of functions has been proposed where functions are classed together if they are related by the following [4,11,12]:

a) negation or inversion of one or more input variables

b) permutation of one or more input variables

c) negation or inversion of the entire function

d) output spectral translation using one of more input variables

e) input spectral translation involving one or more input variables

Lists can then be compiled where the entries are representative or prototype functions from each group. The first three relationships have been used for a classification system known as N.P.N. [4] (standing for negation, permutation, negation) which, for example, requires 221 entries for all functions of $n \leq 4$. Using all five relations has yielded

the Rademacher-Walsh spectral classification which has, for example, only eight entries for $n \leq 4$ as shown in Appendix B. However, it is only convenient to use this system for up to five input variables, where 48 entries are required [13] since for $n > 5$ the lists are too large, for example, for $n \leq 6$ there would have to be in excess of 69,000 entries.

The interesting feature of this classification is that, of the eight entries for $n \leq 4$, seven are threshold functions. This means that a large proportion of non-threshold functions of four variables can be translated into threshold functions, yielding a design strategy, the topology of which is shown in Figure 2.5.

This method can be improved if the remaining non-threshold entries in the classification representative functions are found which are realised using multi-threshold gates. In the case of $n \leq 4$, as stated earlier, there is only one such remaining entry for which the following could be a representative function:

$$f(x) = x_1 \cdot x_2 \oplus x_1 \cdot x_3 \oplus x_1 \cdot x_4 \oplus x_2 \cdot x_3 \oplus x_2 \cdot x_4 \oplus x_3 \cdot x_4$$

which has a spectrum of:

r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
-4	4	4	4	4	4	4	4	4	4	4	-4	-4	-4	-4	-4

Figure 2.6 shows the multi-threshold equivalent of this function, indicating why it is a good choice as a representative since it has unit weights and only two thresholds. If this function is chosen, then for $n \leq 4$ it is possible to realise any function using the topology of Figure 2.5 with a kernal function consisting of a threshold logic gate with at most two thresholds.

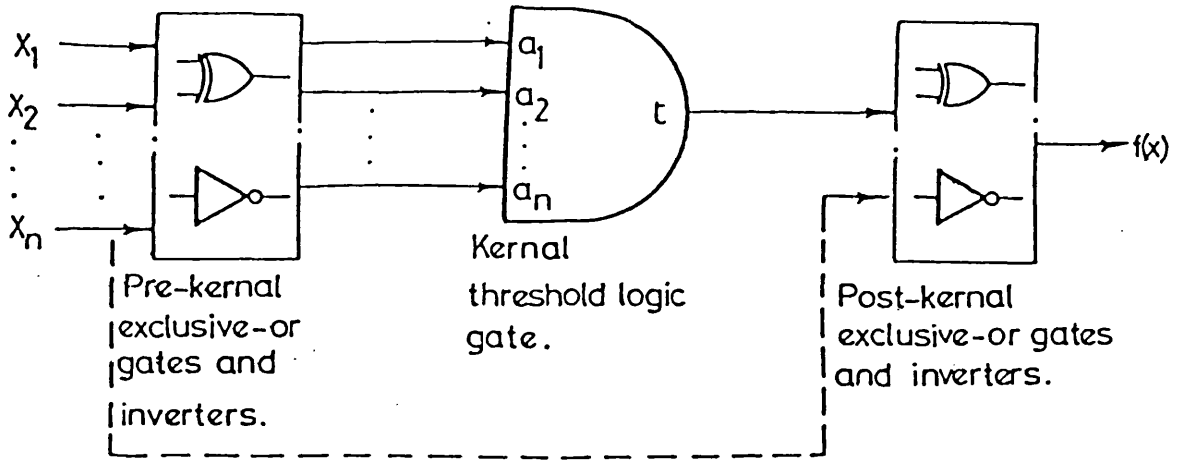


Figure 2.5 Design topology using Rademacher-Walsh spectral classification

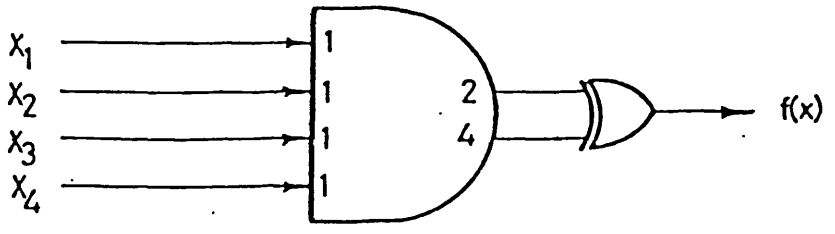


Figure 2.6 Multi-threshold representative function for the single non-threshold class for $n \leq 4$

For $n \leq 5$ there are 27 non-threshold entries, and it is proposed that multi-threshold representative functions may be chosen for these also. However, this is left for future research. For $n > 5$, the listing of representative functions is impractical, and so only functions belonging to groups which have single threshold representative functions can be realised using this method.

2.3 Multi-threshold Logic

A multi-threshold gate [14,15] has more than one output each of which has a threshold associated with it and acts in exactly the same way as a single threshold gate. These outputs are then processed by an exclusive-or gate to give the final output f(x) as in equation 2.10.

$$f(x) = 1 \text{ iff } t_{2j} > \sum_{i=1}^{n} a_i x_i \geq t_{2j-1} \qquad 2.10$$

$$= 0 \text{ otherwise}$$

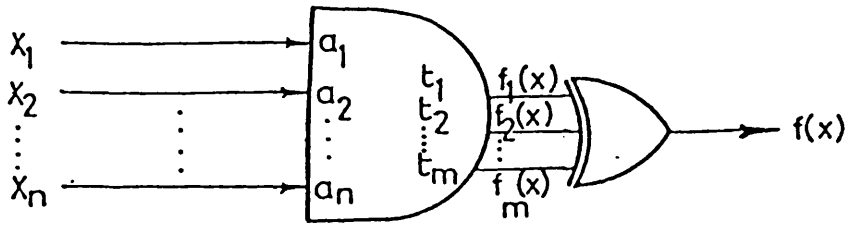
where j = 1,2,3,etc.

Figure 2.7(a) shows the conventional symbol for a multi-threshold logic gate and 2.7(b) shows a graphical representation of its output.

One of the most important aspects of a multi-threshold logic gate is that it is universal, i.e. any function can be realised with one. This is easily shown by considering a function to have weights $a_i = 2^{n-i}$, so that the sum $\sum_{i=1}^{n} a_i x_i$ is different for every minterm position. Thresholds can then be assigned with the values of $\sum_{i=1}^{n} a_i x_i$ where the function output changes from 0 to 1 or 1 to 0. Consider the example given in the previous section:

$$f(x) = x_1 . \bar{x}_2 + \bar{x}_1 . x_2 + x_2 . x_3$$

a)



b)

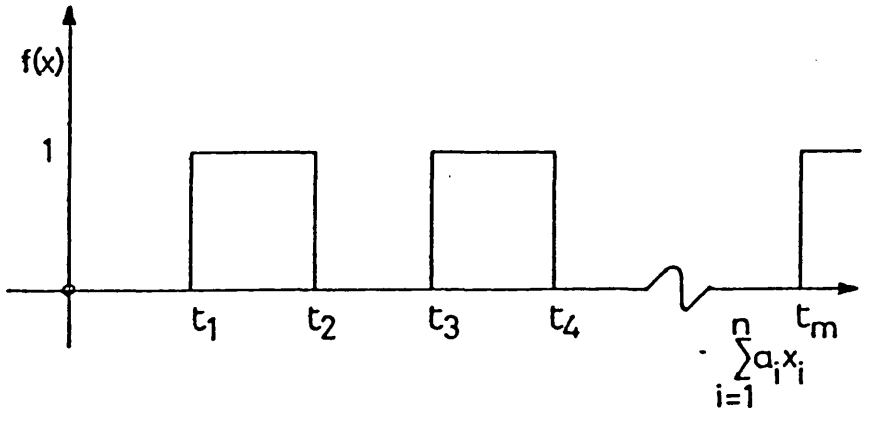


Figure 2.7 (a) Symbol for a multi-threshold logic gate
(b) Multi-threshold logic gate output

Figure 2.8 shows the truth table of this function and the multi-threshold solution. The function changes from 0 to 1 and 1 to 0 at points where $\sum_{i=1}^{n} a_i x_i = 2,6,7$, and so these values are the threshold values.

2^2	2^1	2^0			$t_1=2$	$t_2=6$	$t_3=7$
x_1	x_2	x_3	$\sum_{i=1}^{n} a_i x_i$	f(x)	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	2	1	1	0	0
0	1	1	3	1	1	0	0
1	0	0	4	1	1	0	0
1	0	1	5	1	1	0	0
1	1	0	6	0	1	1	0
1	1	1	7	1	1	1	1

Figure 2.8 Truth table of function $f(x) = x_1 \cdot \overline{x}_2 + \overline{x}_1 \cdot x_2 + x_2 \cdot x_3$ and a multi-threshold solution.

Thus the function is realised with a three threshold gate. Although this weighting procedure could be applied to every function, the values of weights are generally much larger than is necessary; for example, the above function could be realised with a weight set of 2, 2 and 1 instead of 4, 2 and 1.

Another point about multi-threshold logic is that although it is conventional to include an exclusive-or gate at the outputs as in Figure 2.7, a simpler gate could in fact be used. Consider the following:

Since $t_1 < t_2 < t_3 < \ldots < t_m$ 2.11

then $f_1(x) \geq f_2(x) \geq f_3(x) \geq \ldots \geq f_m(x)$ 2.12

This means that it is impossible for f_{i+1} to be a 1 when $f_i(x)$ is a 0, so that a "don't care" term appears in the truth table of Figure 2.9.

$f_i(x)$	$f_{i+1}(x)$	$f(x)$	
0	0	0	
0	1	-	cannot occur
1	0	1	
1	1	0	

Figure 2.9 Truth table of f(x) showing don't care term

Assigning a value of 0 or 1 to the don't care term gives the two alternatives:

$$\text{a)}\quad f(x) = f_i(x).\overline{f_{i+1}(x)} \qquad \text{when the don't care = 0}$$

$$\text{b)}\quad f(x) = f_i(x).\overline{f_{i+1}(x)} + \overline{f_i(x)}.f_{i+1}(x) \qquad \text{when the don't care = 1}$$

$$= f_i(x) \oplus f_{i+1}(x) \qquad 2.13$$

The function (a), known as the "half-exclusive-or" [4], can be used in place of the conventional exclusive-or gate resulting in less gate complexity.

Finally, a single threshold gate has to perform three distinct operations:

i) taking the product of each input variable with its associated weight,

ii) summing all of these products,

iii) comparing the sum with the threshold to give a 0 or a 1 output.

If the first two operations are to be executed it seems wasteful not to extract the most power or usefulness from the gate by having one threshold. Indeed, in some gates such as the Digital-Summation-Threshold-Logic gate [4,12,13] more than one threshold are included in the structure, and so it would seem sensible to incorporate them into the final design.

These three reasons, namely the multi-threshold logic gates universality, the fact that the half-exclusive-or gates can be used instead of the conventional exclusive-or gates, and that the extension from a single threshold to multi-thresholds in a gate is usually available has led to the conclusion that this is the desirable solution to the problem of the realisation of non-threshold functions. However, the difficulty of this solution is the determination of the values of the weights and thresholds. Tables have been compiled [16,17] which list solutions for $n \leq 4$, but beyond this it is impractical to compile tables because they would be enormous.

The previous section showed that spectral translation can be used to find threshold solutions for at least some non-threshold functions of up to eight variables, including all functions of $n \leq 4$. A method is proposed, therefore, to convert from the spectral translation solution to a multi-threshold one.

2.4 Conversion from Spectral Translation Solution to Multi-threshold Solution

As stated earlier, spectral translation covers two distinct operations, output and input, each of which requires a different method of conversion to multi-threshold.

2.4.1 Output Spectral Translation Conversion

2.4.1.1 Single Input Variable

Consider the situation shown in Figure 2.10 where a four input variable threshold function f'(x) is exclusively-ored with the input variable x_1 to give f(x) as in equation 2.14.

$$f(x) = f'(x) \oplus x_1 \qquad 2.14$$

Since f'(x) is a threshold function it can be represented by a weight-threshold vector as in equation 2.15.

$$a'_1\ a'_2\ a'_3\ a'_4;\ t' \qquad 2.15$$

A multi-threshold equivalent of f(x) is shown in Figure 2.11, so that f(x) can be represented by a weight-threshold vector as in equation 2.16 or as an expression of the functions $f_1(x)$ to $f_m(x)$ in equation 2.17.

$$a_1\ a_2\ a_3\ a_4;\ t_1,t_2,\ldots,t_m \qquad 2.16$$

$$f(x) = f_1(x) \oplus f_2(x) \oplus \ldots. \oplus f_m(x) \qquad 2.17$$

The requirement is therefore to find functions $f_1(x), f_2(x)$, etc. which satisfy these two equations. Starting with equation 2.14, if f'(x) is decomposed about the variable x_1 then:

$$f(x) = (\bar{x}_1 g_1(x) \oplus x_1 g_2(x)) \oplus x_1 \qquad 2.18$$

Table 2.1 illustrates this situation.

x_1	f'(x)	x_1
0	$g_1(x)$	0
1	$g_2(x)$	1

Table 2.1 Decomposition of the constituent functions of f(x) about x_1

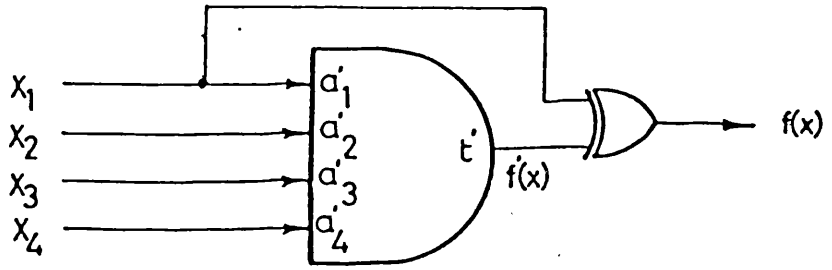


Figure 2.10 Output spectral translation where $f(x) = f'(x) \oplus x_1$

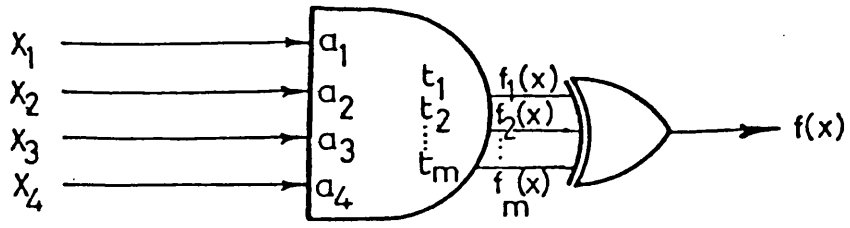


Figure 2.11 Multi-threshold equivalent of f(x)

Clearly, $f'(x)$ and x_1 cannot be a multi-threshold solution because they do not obey equation 2.12. If, however, the terms in Table 2.1 corresponding to $x_1 = 1$, namely $g_2(x)$ and 1, are exchanged then the situation shown in Table 2.2 arises.

x_1	$f_1(x)$	$f_2(x)$
0	$g_1(x)$	0
1	1	$g_2(x)$

Table 2.2 Reordered functions

This gives $f_1(x)$ and $f_2(x)$ as defined in equation 2.19.

$$f_1(x) = f'(x) + x_1$$
$$f_2(x) = f'(x).x_1 \qquad 2.19$$

Since $f_1(x) \geq f_2(x)$ now, equation 2.12 is satisfied. Thus a valid solution has been found if it can be shown that $f_1(x)$ and $f_2(x)$ are threshold functions realisable with the same set of weights but different thresholds.

It can be shown that if a threshold function is decomposed about a variable then the two sub-functions formed are also threshold. Table 2.3 shows $f'(x)$ decomposed about x_1.

x_1	$f'(x)$
0	$a_2' \; a_3' \; a_4' \; ; \; t'$
1	$a_2' \; a_3' \; a_4' \; ; \; t' - a_1'$

Table 2.3 Decomposition of $f'(x)$ about x_1 showing weight-threshold vectors

Now if a new function is considered which has the same weight-threshold vector as f'(x) but which has a constant c added to a'_1 and a second threshold included having a value of the original threshold t' plus c, then its weight-threshold vector is as in equation 2.20 and its decomposition about x_1 shown in Table 2.4.

$$a'_1 + c \; a'_2 \; a'_3 \; a'_4 \; ; \; t', \; t' + c \qquad 2.20$$

x_1	$f_1(x)$	$f_2(x)$
0	$a'_2 \; a'_3 \; a'_4 \; ; \; t'$	$, \; t' + c$
1	$a'_2 \; a'_3 \; a'_4 \; ; \; t' - a'_1 - c$	$, \; t' - a'_1$

Table 2.4 Decomposition of the new function about x_1

If the value of c is chosen such that it is large enough to ensure that the threshold t' + c is never reached by the sum $\sum_{i=2}^{4} a_i x_i$, and that the threshold $t' - a'_1 - c$ is always reached by $\sum_{i=2}^{4} a_i x_i$, then Table 2.4 is equivalent to Table 2.2. Therefore the function represented by equation 2.20 is a solution.

The value of the constant c can be found by the conditions stated above, namely:

$$\sum_{i=2}^{4} a_i x_i < t' + c$$

$$\text{and} \quad \sum_{i=2}^{4} a_i x_i \geq t' - a' - c \qquad 2.21$$

If all the weights are considered positive, which does not affect the generality of the situation, then the maximum and minimum values of $\sum_{i=2}^{4} a_i x_i$ occur when all the x_i's are 1 and 0 respectively, thus:

$$\sum_{i=2}^{4} a_i < t' + c$$

$$0 \geq t' - a_1' - c \qquad 2.22$$

Rearranging and making the number of input variables n for the general case:

$$c \geq \sum_{i=2}^{n} a_i - t' + 1$$

$$c \geq t' - a_1' \qquad 2.23$$

Note that even in the general case, the relevant variable is x_1. This means that given any one variable spectral translation involving x_i, the inputs must be permutated so that $x_i = x_1$. The reason for this becomes clearer later on where more complex general equations will be encountered.

Referring back to the example in section 2.2.1, Figure 2.3, a conversion can now be made, but first the weights have to be made positive. This is done by placing an inverter on any input variable that has a negative weight associated with it; the weight can then be made positive and since the threshold is related to the weights by equation 2.7, it must also be altered to a new value equivalent to its original value minus the weight values of the inputs being inverted. Since these weights were originally negative, the threshold value increases, as is shown in Figure 2.12.

Note that an inverter has been placed on the output of the exclusive-or gate. This is because when the weight a_1' is made positive by placing an inverter on x_1, due to the fact that x_1 is fed forward to the exclusive-or gate, the function is unnecessarily inverted, and hence the re-inversion.

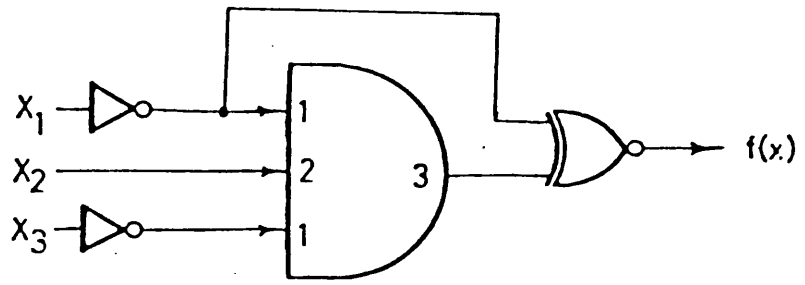


Figure 2.12 Example of output spectral translation to obtain the function $f(x) = x_1 \cdot \bar{x}_2 + \bar{x}_1 \cdot x_2 + x_2 \cdot x_3$

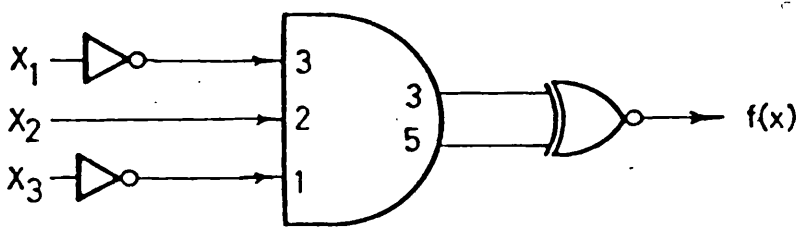


Figure 2.13 Multi-threshold solution of $f(x) = x_1 \cdot \bar{x}_2 + \bar{x}_1 \cdot x_2 + x_2 \cdot x_3$

Equation 2.23 can now be used to find the value of c.

$$c \geq 4 - 3 + 1 = 2$$

$$c \geq 3 - 1 = 2$$

$$\therefore \quad c = 2$$

Equation 2.20 gives the weight-threshold vector as:

$$a_1 \; a_2 \; a_3 \quad t_1 \; t_2$$
$$3 \quad 2 \quad 1 \quad ; \; 3, 5$$

Thus the final realisation is shown in Figure 2.13.

2.4.1.2 More than One Input Variable

The technique described in the previous section of taking the weight-threshold vector of the function f'(x) and increasing the relevant weight by the constant c and adding a threshold can also be applied if more than one variable is involved. Consider the situation where two input variables are exclusively-ored with the function as in equation 2.24.

$$f(x) = f'(x) \oplus x_1 \oplus x_2 \qquad 2.24$$

If the constituents of f(x) are now decomposed about the variables x_1 and x_2 as in Table 2.5, it is immediately clear that as before these three functions do not obey equation 2.12.

$x_1 x_2$	f'(x)	x_1	x_2
0 0	$g_1(x)$	0	0
0 1	$g_2(x)$	0	1
1 0	$g_3(x)$	1	0
1 1	$g_4(x)$	1	1

Table 2.5 Decomposition of the constituents of f(x) about x_1 and x_2

If, as before, the terms are reorganised so that they do obey equation 2.12 then the result is shown in Table 2.6, and the functions $f_1(x)$, $f_2(x)$ and $f_3(x)$ are as in equation 2.25.

x_1x_2	$f_1(x)$	$f_2(x)$	$f_3(x)$
0 0	$g_1(x)$	0	0
0 1	1	$g_2(x)$	0
1 0	1	$g_3(x)$	0
1 1	1	1	$g_4(x)$

Table 2.6 Reorganised decomposition of the constituents of f(x)

$$f_1(x) = f'(x) + x_1 + x_2$$

$$f_2(x) = f'(x).x_1 + f'(x).x_2 + x_1.x_2 \qquad 2.25$$

$$f_3(x) = f'(x).x_1.x_2$$

The requirement is, therefore, to show that $f_1(x)$, $f_2(x)$ and $f_3(x)$ are threshold functions which can be realised with the same set of weights but different thresholds.

Consider the weight-threshold vector of equation 2.26,

$$a_1' + c\ a_2' + c\ a_3'\ a_4' \ ;\ t',\ t' + c,\ t' + 2c \qquad 2.26$$

Decomposed about x_1 and x_2 gives the situation shown in Table 2.7.

x_1x_2	$f_1(x)$	$f_2(x)$	$f_3(x)$
0 0	$a_3' a_4'$; t'	, $t' + c$	, $t' + 2c$
0 1	$a_3' a_4'$; $t' - a_2' - c$	, $t' - a_2'$	, $t' - a_2' + c$
1 0	$a_3' a_4'$; $t' - a_1' - c$	, $t' - a_1'$	, $t' - a_1' + c$
1 1	$a_3' a_4'$; $t' - a_1' - a_2' - 2c$	, $t' - a_1' - a_2' - c$	, $t' - a_1' - a_2'$

Table 2.7 Decomposition of the function given in 2.26 about x_1 and x_2

If c is chosen such that all functions with thresholds containing the term $-c$ are equivalent to logic 1's, and all those with thresholds containing the term $+c$ are equivalent to logic 0's, then Tables 2.7 and 2.6 are identical. In order to achieve this, c must obey all of the following conditions:

$$\begin{aligned} c &\geq t' - a_1' \\ c &\geq t' - a_2' \\ c &\geq a_1' + a_3' + a_4' + 1 - t' \\ c &\geq a_2' + a_3' + a_4' + 1 - t' \end{aligned} \qquad 2.27$$

Previously, it was stated that if an output spectral translation involves only one input variable, then that variable is to be called x_1. Similarly, if two input variables are involved then they are to be called x_1 and x_2, and have associated weights a_1' and a_2' which satisfy the condition:

$$a_1' \geq a_2' \qquad 2.28$$

More generally, if k variables are involved, then they are to be called x_1 to x_k where their associated weights a_1' to a_k' satisfy the condition:

$$a'_1 > a'_2 > a'_3 > .... > a'_k \qquad 2.29$$

Thus the equation for a general conversion is:

$$\begin{aligned} a_i &= a'_i + c \qquad \text{for } i = 1 \text{ to } k \\ a_i &= a'_i \qquad \text{for } i = k+1 \text{ to } n \\ t_j &= t' + (j-1)c \qquad \text{for } j = 1 \text{ to } k+1 \end{aligned} \qquad 2.30$$

and where $c \geq t' - a'_k$

and $c \geq \sum_{i=1}^{n} a'_i + 1 - t' - a'_k$

2.4.1.3 Non-threshold Initial Function f'(x)

Consider the case where the initial function f'(x) is itself a non-threshold function, but which can be represented by a weight-threshold vector with p thresholds, where p > 1, as in equation 2.31.

$$a'_1\ a'_2\ a'_3\ a'_4\ ;\ t'_1, t'_2, ..., t'_p \qquad 2.31$$

For instance, let p = 2, and exclusively-or the function with x_1 to get f(x) as in equation 2.32.

$$f(x) = f'(x) \oplus x_1 = f'_1(x) \oplus f'_2(x) \oplus x_1 \qquad 2.32$$

In order to convert this to a completely multi-threshold solution each function $f'_i(x)$ has to be treated separately, as in the previous section. Thus each function is exclusively-ored with x_1 as in equation 2.33.

$$f(x) = f'_1(x) \oplus f'_2(x) \oplus x_1 = (f'_1(x) \oplus x_1) \oplus (f'_2(x) \oplus x_1) \oplus x_1 \qquad 2.33$$

The extra x_1 terms do not alter the function f(x) since $x_1 \oplus x_1 = 0$. Table 2.8 illustrates the functions decomposed about x_1.

x_1	$f_1'(x)$	x_1	$f_2'(x)$	x_1	x_1
0	$g_1(x)$	0	$h_1(x)$	0	0
1	$g_2(x)$	1	$h_2(x)$	1	1

Table 2.8 Decomposition of the constituent functions of f(x)

The same method as described in section 2.4.1.1 can be applied to each function, resulting firstly in the reordered functions as in Table 2.9, a weight-threshold vector as in equation 2.34 and the decomposition of this function in Table 2.10.

x_1	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$
0	$g_1(x)$	$h_1(x)$	0	0	0
1	1	1	1	$g_2(x)$	$h_2(x)$

Table 2.9 Reordered decomposition

$$a_1' + c\ a_2'\ a_3'\ a_4'\ ;\ t_1', t_2', c, t_1' + c,\ t_2' + c \qquad 2.34$$

x_1	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$
0	$a_2'\ a_3'\ a_4'\ ;\ t_1'$	t_2'	c	$t_1' + c$	$t_2' + c$
1	$a_2'\ a_3'\ a_4'\ ;\ t_1' - a_1' - c$	$t_2' - a_1' - c$	$-a_1'$	$t_1' - a_1'$	$t_2' - a_1'$

Table 2.10 Decomposition of the function of equation 2.34 about x_1

If c is chosen as before so that thresholds containing -c belong to functions which are equivalent to logic 1's, and those with +c are equivalent to logic 0's, then Tables 2.9 and 2.10 are identical. Note that a new function has been added, namely x_1, which has a threshold of c. This means that the value of c must be such that:

$$c \geq a'_2 + a'_3 + a'_4 + 1 \qquad 2.35$$

$$c \geq t'_2 - a'_1$$

This extra function appears because initially f'(x) consisted of an even number of functions, i.e. p was even. If k variables are involved in the translation then k extra functions are required if p is even, having threshold values of c, 2c, 3c,..., kc. This can be summarised into a general equation with the thresholds ordered so that the extra ones appear last, thus:

$$a'_i = a'_i + c \qquad \text{for } i = 1 \text{ to } k$$

$$a_i = a'_i \qquad \text{for } i = k + 1 \text{ to } n$$

$$t_{j+(q-1)(k+1)} = t'_q + (j - 1)c \qquad \text{for } j = 1 \text{ to } k + 1,\ q = 1 \text{ to } p$$

and if p is even: $t_{j+p(k+1)} = jc \qquad \text{for } j = 1 \text{ to } k \qquad 2.36$

where $c \geq t'_p - a'_k$

and $$c \geq \sum_{i=k+1}^{n} a'_i + 1 - \left[\frac{1 - (-1)^p}{2}\right] \cdot \left[\sum_{i=1}^{k} a'_i - t'_1\right]$$

2.4.1.4 Final Adjustments

A situation which has not yet been taken into account is where some of the functions in the multi-threshold solution are identical and therefore can be cancelled out. For example, in the Tables 2.9 and 2.10, if $h_1(x) = 0$, then the functions $f_2(x)$ and $f_3(x)$ are identical, but one has a threshold of t'_2 and the other a threshold of c, and so from the thresholds alone it may not be immediately apparent that the functions are the same. One way to overcome this problem is as follows. Firstly, note that the value of c in equation 2.35 has to be greater than $a'_2 + a'_3 + a'_4 + 1$, so that the function $f_3(x)$ in Table 2.10 is equivalent to x_1. If the threshold t_3 is raised to $c + a'_1$, then the

function $f_3(x)$ is unaltered if the value of c obeys the following:

$$c \geq a'_2 + a'_3 + a'_4 + 1 - a'_1 \qquad 2.37$$

Clearly this is less than that required in equation 2.35, and therefore it is an improvement. Reconsidering all five functions, the value of c must be such that:

$$\begin{aligned} \text{for } f_1(x) &: \quad c \geq t'_1 - a'_1 \\ f_2(x) &: \quad c \geq t'_2 - a'_1 \\ f_3(x) &: \quad c \geq a'_2 + a'_3 + a'_4 + 1 - a'_1 \qquad 2.38 \\ f_4(x) &: \quad c \geq a'_2 + a'_3 + a'_4 + 1 - t'_1 \\ \text{and } f_5(x) &: \quad c \geq a'_2 + a'_3 + a'_4 + 1 - t'_2 \end{aligned}$$

Let $c = a'_2 + a'_3 + a'_4 + 1 - a'_1 \qquad 2.39$

Then for $f_1(x)$: $a'_2 + a'_3 + a'_4 + 1 - a'_1 \geq t'_1 - a'_1$

or $t'_1 \leq a'_2 + a'_3 + a'_4 + 1 \qquad 2.40$

otherwise $g_1(x) = 0$ from Table 2.10.

If $g_1(x) = 0$, then $f_1(x) = f_3(x) = x_1$, and so t_1 must be altered to t_3, i.e. $c + a'_1$.

The same applies to $f_2(x)$.

for $f_4(x)$: $a'_2 + a'_3 + a'_4 + 1 - a'_1 \geq a'_2 + a'_3 + a'_4 + 1 - t'_1$

or $t'_1 \geq a'_1 \qquad 2.41$

otherwise $g_2(x) = 1$ from Table 2.10.

If $g_2(x) = 1$, then $f_4(x) = f_3(x) = x_1$, and so t_4 must be altered to t_3, i.e. $c + a'_1$.

The same applies to $f_5(x)$.

Thus the value of c in equation 2.39 is valid if the thresholds are:

$$t_1 = t_1' + (c + a_1' - t_1') \quad \text{include terms in brackets if } t_1' \geq a_2' + a_3' + a_4' + 1$$

$$t_2 = t_2' + (c + a_1' - t_2') \quad \text{" " " " " } t_2' \geq a_2' + a_3' + a_4' + 1$$

$$t_3 = c + a_1'$$

$$t_4 = t_1' + c + (a_1' - t_1') \quad \text{" " " " " } t_1' \leq a_1'$$

$$t_5 = t_2' + c + (a_1' - t_2') \qquad t_2' \leq a_1'$$

2.42

The presentation of these thresholds can be made more formal if the following function is included:

$$F = T(A \geq B)$$

where $F = 0$ iff $A < B$ 2.43

and $F = 1$ iff $A \geq B$

Figure 2.14 shows the case when A and B have values of up to two.

A \ B	0	1	2
0	1	0	0
1	1	1	0
2	1	1	1

Figure 2.14 $F = T\ (A \geq B)$

Using this function results, for example, in t_1 from equation 2.42 becoming:

$$t_1 = t_1' + T(t_1' \geq a_2' + a_3' + a_4' + 1).\left[c + a_1' - t_1'\right] \qquad 2.44$$

A general solution can now be given which is the extension of equation 2.36 but which now has a new value for c, as in equation 2.39, and which incorporates the adjustments to the thresholds using the above method.

$$a_i = a'_i + c \qquad \text{for } i = 1 \text{ to } k$$

$$a_i = a'_i \qquad \text{for } i = k+1 \text{ to } n \qquad 2.45$$

$$t_{j+(q-1)(k+1)} = t'_q + (j-1)c + T\left(t'_q \geq \sum_{i=k+1}^{n} a'_i + \sum_{i=1}^{j} a'_i - a'_j + 1\right).$$

$$\left[c - t'_q + \sum_{i=k+1-j}^{k} a'_i - T(j \geq k+1).\left[a'_0\right]\right]$$

$$+ T\left(\sum_{i=k+1-j}^{k} a'_i - a'_{k+1-j} \geq t'_q\right).\left[\sum_{i=k+1-j}^{k} a'_i - t'_q - a'_{k+1-j}\right]$$

$$\text{for } j = 1 \text{ to } k+1,\ q = 1 \text{ to } p$$

If p is even then:

$$t_{p(k+1)+j} = jc + \sum_{i=k+1-j}^{k} a'_i \qquad \text{for } j = 1 \text{ to } k$$

where:

$$c = \sum_{i=1}^{n} a'_i - \sum_{i=2k+1+(-1)^{k+1}/4}^{k} a'_i - \sum_{i=2k+3+(-1)^{k}/4}^{k} a'_i + 1$$

Although this equation can be applied easily with the aid of a computer, by hand it is difficult to use. However, the method described in section 2.3 and shown in Figure 2.8 for proving the universality of multi-threshold logic can be adopted with the weights from equation 2.45 being used. Both methods are shown in the following example.

Example

$$f(x) = \overline{\bar{x}_1 . x_2 . x_3 + \bar{x}_1 . \bar{x}_2 . \bar{x}_3 + x_1 . \bar{x}_2 . x_3 + \bar{x}_2 . \bar{x}_3 . \bar{x}_4 + x_1 . x_3 . \bar{x}_4}$$

The spectrum of this function is:

r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
0	0	4	-4	4	-4	4	-4	8	0	0	8	0	0	4	-4

Comparing the first five coefficients with the tables of Chow's parameters shows that there is no entry for $n \leq 4$ which is 44000, and so the function is non-threshold. However, if the function is exclusively-ored with x_2 and x_3, then output spectral translation results in the spectrum:

r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
8	8	-4	4	4	4	-4	-4	0	0	0	0	0	0	4	-4

The tables of Chow's parameters have an entry of 88444; therefore this function is a threshold function, and the corresponding a parameters are 22111. Thus:

a_0	a_1	a_2	a_3	a_4
2	2	-1	1	1

Using equation 2.7 gives:

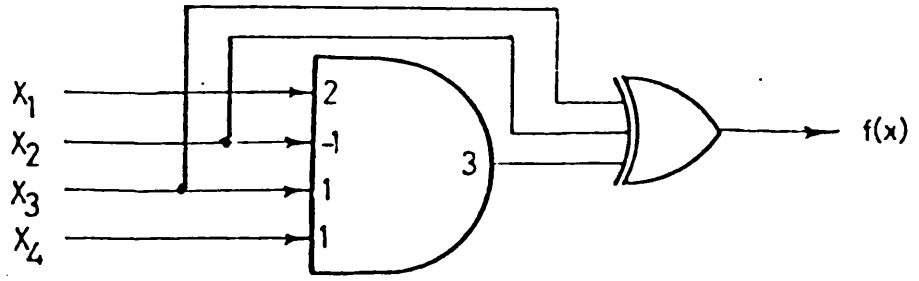
$$t = 3$$

Thus the weight-threshold vector of f(x) is:

$$2\ -1\ 1\ 1\ 1\ ;\ 3$$

Figure 2.15(a) shows the final realisation of f(x), and Figure 2.15(b) shows the initial adjustments which have to be made in order that a conversion to multi-threshold can occur, namely the weights must be made positive, and the relevant input variables which are involved in the translation must be ordered according to equation 2.29.

a)



b)

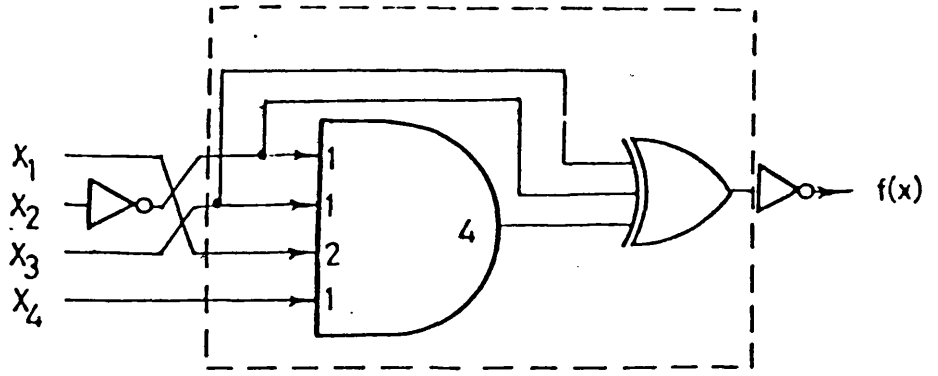


Figure 2.15 (a) Solution to function f(x)
(b) Reorganised function f(x) with positive weights

The function shown inside the dotted line can now be converted to multi-threshold using equation 2.45.

In this example k = 2, n = 4.

$$c = \sum_{i=1}^{4} a'_i - \sum_{i=1}^{2} a'_i - \sum_{i=2}^{2} a'_i + 1 = a'_3 + a'_4 - a'_2 + 1 = 3$$

Thus, $a_1 = 4$, $a_2 = 4$, $a_3 = 2$, $a_4 = 1$.

The thresholds according to equation 2.45 are therefore:

k = 2, n = 4, q = 1, c = 3, t' = 4.

$$t_j = 4 + (j-1).3 + T(4 \geq 3 + \sum_{i=1}^{j} a'_i - a'_{j+1}).\left[3 - 4 + \sum_{i=3-j}^{2} a'_i - T(j \geq 3).\left[a'_0\right]\right] + T(\sum_{i=3-j}^{2} a'_i - a'_{3-j} \geq 4).\left[\sum_{i=3-j}^{2} a'_i - 4 - a'_{3-j}\right]$$

for j = 1 to 3

$$t_1 = 4 + T(4 \geq 4).\left[-1 + 1\right] + T(0 \geq 4).\left[-4\right] = 4$$

$$t_2 = 4 + 3 + T(4 \geq 4 + 1).\left[-1 + 2\right] + T(1 \geq 4).\left[1 - 4\right] = 7$$

$$t_3 = 4 + 6 + T(4 \geq 4 + 2).\left[-1 + 2\right] + T(2 \geq 4).\left[2 - 4\right] = 10$$

The solution is shown in Figure 2.16.

The thresholds could also have been calculated by using $\sum_{i=1}^{n} a_i x_i$ with the a_i values of 2,4,4,1 calculated as before, and comparing with the function $\overline{f(x_1,x_2,x_3,x_4)} = f''(x)$ as in Figure 2.17.

Thus comparing $\sum_{i=1}^{n} a_i x_i$ with f"(x) in Figure 2.17(a) shows that f"(x) changes from 0 to 1 at values of 4 and 10, and 1 to 0 at the value of 7, which can be represented as in Figure 2.17(b).

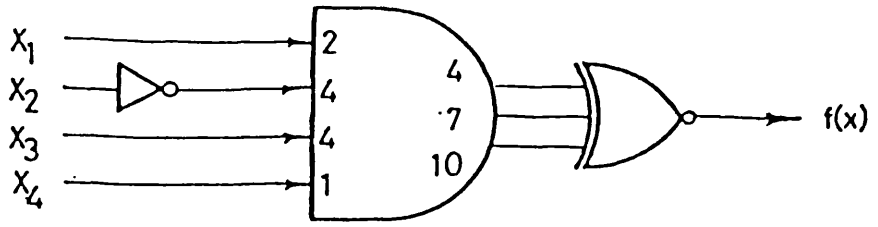


Figure 2.16 Multi-threshold solution of f(x)

a)

x_1	x_2	x_3	x_4	$\sum_{i=1}^{n} a_i x_i$	f"(x)
	2	4	4	1	
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	4	1
0	0	1	1	5	1
0	1	0	0	4	1
0	1	0	1	5	1
0	1	1	0	8	0
0	1	1	1	9	0
1	0	0	0	2	0
1	0	0	1	3	0
1	0	1	0	6	1
1	0	1	1	7	0
1	1	0	0	6	1
1	1	0	1	7	0
1	1	1	0	10	1
1	1	1	1	11	1

b)

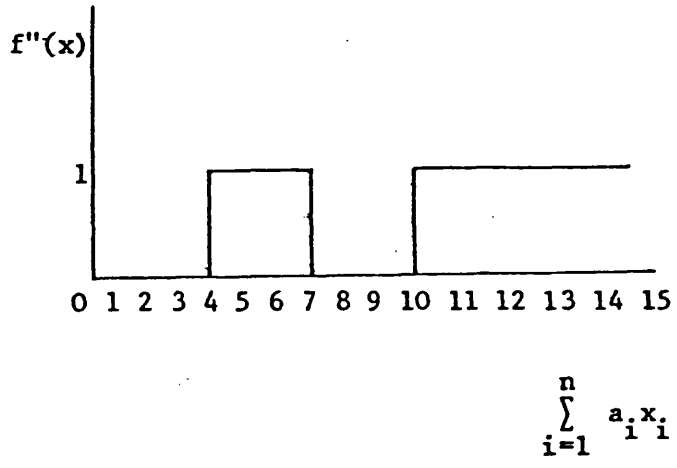


Figure 2.17 (a) Truth table of function f"(x)

(b) Graphical illustration of multi-threshold solution of f"(x)

2.4.2 Input Spectral Translation Conversion

2.4.2.1 Single Input Variable

Consider the situation shown in Figure 2.18 of a four input variable threshold function f'(x) where its input variable x_1 has been replaced by the exclusive-or function of x_1 and x_2 as in equation 2.46.

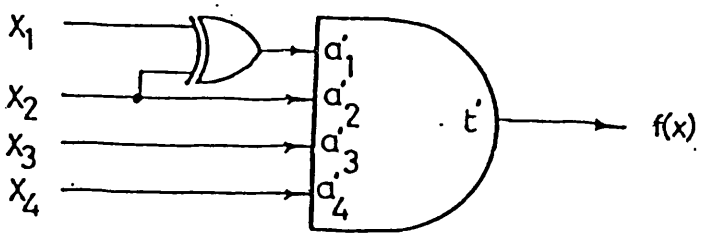


Figure 2.18 Input spectral translation where x_1 is replaced by $x_1 \oplus x_2$, to give f(x)

$$f(x_1,x_2,x_3,x_4) = f'(x_1 \oplus x_2,x_2,x_3,x_4) \qquad 2.46$$

As in section 2.4.2.2, since f'(x) is a threshold function it can be represented by the weight-threshold vector of equation 2.15, and the multi-threshold equivalent function can be represented as in Figure 2.11 and equations 2.16 and 2.17. The requirement is therefore to find a set of functions $f_1(x)$ to $f_m(x)$ which satisfy equations 2.16 and 2.17 and which, when exclusively-ored, give the function f(x).

If f'(x) is decomposed about the variables x_1 and x_2 then:

$$f'(x) = \bar{x}_1.\bar{x}_2.g_1(x) + \bar{x}_1.x_2.g_2(x) + x_1.\bar{x}_2.g_3(x) + x_1.x_2.g_4(x) \qquad 2.47$$

Substituting $x_1 \oplus x_2$ for x_1 to give f(x):

$$f(x) = \bar{x}_1.\bar{x}_2.g_1(x) + \bar{x}_1.x_2.g_4(x) + x_1.\bar{x}_2.g_3(x) + x_1.x_2.g_2(x) \qquad 2.48$$

This is illustrated in Table 2.11

x_1 x_2	f'(x)	f(x)
0 0	$g_1(x)$	$g_1(x)$
0 1	$g_2(x)$	$g_4(x)$
1 0	$g_3(x)$	$g_3(x)$
1 1	$g_4(x)$	$g_2(x)$

Table 2.11 Decomposition of f'(x) and f(x) about x_1 and x_2

Now consider the set of functions given in equation 2.49.

$$
\begin{aligned}
f_1(x) &= g_1(x) + x_1 + x_2 \\
f_2(x) &= x_1 + x_2 \\
f_3(x) &= x_1 . g_3(x) + x_2 . g_4(x) + x_1 . x_2 \\
f_4(x) &= x_1 . x_2 \\
f_5(x) &= x_1 . x_2 . g_2(x)
\end{aligned}
\qquad 2.49
$$

Decomposing each of these functions about x_1 and x_2 gives:

x_1x_2	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$
0 0	$g_1(x)$	0	0	0	0
0 1	1	1	$g_4(x)$	0	0
1 0	1	1	$g_3(x)$	0	0
1 1	1	1	1	1	$g_2(x)$

Table 2.12 Decomposition of the function $f_1(x)$ to $f_5(x)$

It can be seen from Table 2.12 that if the five functions are exclusively-ored then the result is f(x), and thus these functions represent a solution which obeys equation 2.12. Thus, if these functions are threshold and can be represented by the same weights but different thresholds then a multi-threshold solution has been found.

The decomposition of the function, represented by the weight-threshold vector of equation 2.50, about x_1 and x_2 is shown in Table 2.13.

$$c\ a_2' + c\ a_3'\ a_4' \ ;\ t', c, t' + c - a_1', 2c + a_2', t' + 2c \qquad 2.50$$

$x_1 x_2$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$
0 0	$a_3' a_4'$; t'	c	$t' + c - a_1'$	$2c + a_2'$	$t' + 2c$
0 1	$a_3' a_4'$; $t' - a_2' - c$	$-a_2'$	$t' - a_1' - a_2'$	c	$t' - a_2' + c$
1 0	$a_3' a_4'$; $t' - c$	0	$t' - a_1'$	$c + a_2'$	$t' + c$
1 1	$a_3' a_4'$; $t' - a' - 2c$	$-a_2' - c$	$t' - a_1' - a_2' - c$	0	$t' - a_2'$

Table 2.13 Decomposition of weight-threshold vector of equation 2.50 about x_1 and x_2

If in Table 2.13 all weights are considered positive, then all functions with thresholds which are negative, 0, or contain -c can be considered logic 1's, and those containing +c as logic 0's so that Table 2.13 is equivalent to Table 2.12. The value of c must be chosen, therefore, such that all the following conditions are obeyed:

$$\begin{aligned} &\text{for } f_1(x): \ c \geq t' \\ &\quad f_2(x): \ c \geq a_3' + a_4' + 1 \\ &\quad f_3(x): \ c \geq a_1' + a_3' + a_4' + 1 - t' \ \text{ and } \ c \geq t' - a_1' - a_2' \\ &\quad f_4(x): \ c \geq a_3' + a_4' + 1 \\ &\text{and } f_5(x): \ c \geq a_2' + a_3' + a_4' + 1 - t' \end{aligned} \qquad 2.51$$

Let $c = a_3' + a_4' + 1$ 2.52

which immediately satisfies $f_2(x)$ and $f_4(x)$.

Then for $f_1(x)$: $a_3' + a_4' + 1 \geq t'$ 2.53

otherwise $g_1(x) = 0$ from Table 2.12.

If $g_1(x) = 0$, then $f_1(x) = f_2(x)$, and so t_1 can be altered to t_2, i.e. c.

For $f_3(x)$: $a'_3 + a'_4 + 1 \geq a'_1 + a'_3 + a'_4 + 1 - t'$

or $t' \geq a'_1$ 2.54

otherwise $g_4(x) = g_3(x) = 1$ from Table 2.12.

If $g_4(x) = g_3(x) = 1$, then $f_3(x) = f_2(x)$ and so t_3 can be altered to t_2, i.e. c.

Also: $a'_3 + a'_4 + 1 \geq t' - a'_1 \tau a'_2$

or $a'_1 + a'_2 + a'_3 + a'_4 + 1 \geq t'$ 2.55

otherwise $g_4(x) = g_3(x) = 0$ from Table 2.12.

If $g_4(x) = g_3(x) = 0$, then $f_3(x) = f_4(x)$ and so t_3 can be altered to t_4, i.e. $2c + a'_2$.

For $f_5(x)$: $a'_3 + a'_4 + 1 \geq a'_2 + a'_3 + a'_4 + 1 - t'$

or $t' \geq a'_2$ 2.56

otherwise $g_2(x) = 1$ from Table 2.12.

If $g_2(x) = 1$, then $f_5(x) = f_4(x)$ and so t_5 can be altered to t_4, i.e. $2c + a'_2$.

Thus c in equation 2.52 is valid if the equations for the thresholds are as follows, where $T(A \geq B)$ is as before in equation 2.43:

$$
\begin{aligned}
t_1 &= t' + T(t' \geq a'_3 + a'_4 + 1).\left[c - t'\right] \\
t_2 &= c \\
t_3 &= t' + c - a'_1 + T(a'_1 \geq t').\left[a'_1 - t'\right] + T(t' \geq a'_1 + a'_2 + a'_3 + a'_4 + 1). \\
&\qquad \left[c + a'_1 + a'_2 - t'\right] \\
t_4 &= 2c + a'_2 \\
t_5 &= t' + 2c + T(a'_2 \geq t').\left[a'_2 - t'\right] \qquad 2.57
\end{aligned}
$$

The general solution for n input variables is therefore:

$$t_1 = t' + T(t' \geq \sum_{i=3}^{n} a'_i + 1).\left[c - t'\right]$$

$$t_2 = c$$

$$t_3 = t' + c - a'_1 + T(a'_1 \geq t').\left[a'_1 - t'\right] + T(t' \geq \sum_{i=1}^{n} a'_i + 1).\left[c + a'_1 + a'_2 - t'\right] \qquad 2.58$$

$$t_4 = 2c + a'_2$$

$$t_5 = t' + 2c + T(a'_2 \geq t').\left[a'_2 - t'\right]$$

where $c = \sum_{i=3}^{n} a'_i + 1$

Note that the general solution applies only to situations where the variable x_1 is replaced by $x_1 \oplus x_2$, so that if a function has an input variable x_i replaced by $x_i \oplus x_j$, its inputs must be reordered such that i = 1, j = 2.

2.4.2.2 Non-threshold Initial Function f'(x)

This situation is identical to that at the start of section 2.4.1.3, so that f'(x) is still as in equation 2.31. Again, in order to find the multi-threshold solution to f(x), each of the functions $f'_i(x)$ are treated separately, and so in this case each would be expanded into five threshold functions according to equation 2.58. For example, consider a two threshold function f'(x) as in equation 2.59.

$$a'_1\ a'_2\ a'_3\ a'_4\ ;\ t'_1, t'_2 \qquad 2.59$$

Using equation 2.58, the five thresholds for each new function are:

a) $t_1 = t'_1 + T(t'_1 \geq \sum_{i=3}^{n} a'_i + 1).\ \left[c - t'_1\right]$

$t_2 = c$

$t_3 = t'_1 + c - a'_1 + T(a'_1 \geq t'_1).\left[a'_1 - t'_1\right] + T(t'_1 \geq \sum_{i=1}^{n} a'_i + 1).$

$\left[c + a'_1 + a'_2 - t'_1\right]$

$t_4 = 2c + a'_2$

$t_5 = t'_1 + 2c + T(a'_2 \geq t'_1).\ \left[a'_2 - t'_1\right]$

b) $t_1 = t'_2 + T(t'_2 \geq \sum_{i=3}^{n} a'_i + 1).\ \left[c - t'_2\right]$ 2.60

$t_2 = c$

$t_3 = t'_2 + c - a'_1 + T(a'_1 \geq t'_2).\ \left[a'_1 - t'_2\right] + T(t'_2 \geq \sum_{i=1}^{n} a'_i + 1).$

$\left[c + a'_1 + a'_2 - t'_2\right]$

$t_4 = 2c + a'_2$

$t_5 = t'_2 + 2c + T(a'_2 \geq t'_2).\ \left[a'_2 - t'_2\right]$

Clearly each expansion of five thresholds contains t_2 and t_4 which are independent of the original threshold and pairs of them can be cancelled out. Thus if the original function f'(x) has p thresholds, then each one expands into three new thresholds which are obtained as t_1, t_3 and t_5 in equation 2.58, and if p is odd then two more are added which are equivalent to t_2 and t_4.

2.4.2.3 More than One Input Variable

In this case, an input variable is replaced by the exclusive-or function of the same input variable and more than one other. Consider the situation of the function in equation 2.61.

$$f(x_1,x_2,x_3,x_4) = f'(x_1 \oplus x_2 \oplus x_3,x_2,x_3,x_4) \qquad 2.61$$

Table 2.14 shows the decomposition of this function f'(x) about x_1,x_2 and x_3. Also shown is a set of seven functions which when exclusively-ored with each other give the function f(x).

$x_1x_2x_3$	f'(x)	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_7(x)$	f(x)
0 0 0	$g_1(x)$	$g_1(x)$	0	0	0	0	0	0	$g_1(x)$
0 0 1	$g_2(x)$	1	1	$g_6(x)$	0	0	0	0	$g_6(x)$
0 1 0	$g_3(x)$	1	1	$g_7(x)$	0	0	0	0	$g_7(x)$
0 1 1	$g_4(x)$	1	1	1	1	$g_4(x)$	0	0	$g_4(x)$
1 0 0	$g_5(x)$	1	1	$g_5(x)$	0	0	0	0	$g_5(x)$
1 0 1	$g_6(x)$	1	1	1	1	$g_2(x)$	0	0	$g_2(x)$
1 1 0	$g_7(x)$	1	1	1	1	$g_3(x)$	0	0	$g_3(x)$
1 1 1	$g_8(x)$	1	1	1	1	1	1	$g_8(x)$	$g_8(x)$

Table 2.14 Decomposition of f(x), f'(x) and $f_1(x)$ to $f_7(x)$, about x_1, x_2 and x_3

The functions $f_1(x)$ to $f_7(x)$ satisfy equation 2.12, and so it is required to show that they are threshold functions which can be realised with the same set of weights but different thresholds to obtain a multi-threshold solution.

Table 2.15 shows the decomposition of the function, represented by the weight-threshold vector of equation 2.62, about the variables x_1, x_2 and x_3.

$x_1x_2x_3$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_7(x)$
0 0 0	a_4'; t'	c	$t' + c - a_1'$	$2c + a_3'$	$t' + 2c$	$3c + a_2' + a_3'$	$t' + 3c - a_1'$
0 0 1	a_4'; $t' - a_3' - c$	$-a_3'$	$t' - a_1' - a_3'$	c	$t' - a_3' + c$	$2c + a_2'$	$t' - a_1' - a_3' + 2c$
0 1 0	a_4'; $t' - a_2' - c$	$-a_2'$	$t' - a_1' - a_2'$	$c + a_3' - a_2'$	$t' - a_2' + c$	$2c + a_3'$	$t' - a_1' - a_2' + 2c$
0 1 1	a_4'; $t' - a_2' - a_3' - 2c$	$-a_2' - a_3' - c$	$t' - a_1' - a_2' - a_3' - c$	$-a_2'$	$t' - a_2' - a_3'$	c	$t' - a_1' - a_2' - a_3' + c$
1 0 0	a_4'; $t' - c$	0	$t' - a_1'$	$c + a_3'$	$t' + c$	$2c + a_2' + a_3'$	$t' - a_1' + c$
1 0 1	a_4'; $t' - a_3' - 2c$	$-a_3' - c$	$t' - a_1' - a_3' - c$	0	$t' - a_3'$	$c + a_2'$	$t' - a_1' - a_3' + c$
1 1 0	a_4'; $t' - a_2' - 2c$	$-a_2' - c$	$t' - a_1' - a_2' - c$	$a_3' - a_2'$	$t' - a_2'$	$c + a_3'$	$t' - a_1' - a_2' + c$
1 1 1	a_4'; $t' - a_2' - a_3' - 3c$	$-a_2' - a_3' - 2c$	$t' - a_1' - a_2' - a_3' - 2c$	$-a_2' - c$	$t' - a_2' - a_3' - c$	0	$t' - a_1' - a_2' - a_3'$

Table 2.15 Decomposition of the function given in equation 2.60 about the variables x_1, x_2 and x_3

$$c\ a'_2 + c\ a'_3 + c\ a'_4\ ;\ t', c, t' + c - a'_1,\ 2c + a'_3,\ t' + 2c,\ 3c + a'_2 + a'_3,\ t' + 3c - a'_1 \qquad 2.62$$

If in Table 2.15 all the weights are positive and the involved input variables x_2 and x_3 are arranged such that $a'_2 > a'_3$, then if c is large enough, Tables 2.15 and 2.14 are identical. The actual value c can be shown to be as in equation 2.63, using the same arguments as in equations 2.52 to 2.56 in section 2.4.2.1:

$$c = a'_2 - a'_3 + a'_4 + 1 \qquad 2.63$$

if the thresholds include conditional adjustments as in equation 2.64.

$$\begin{aligned}
t_1 &= t' + T(t' \geq a'_4 + 1).\ \left[c - t'\right] \\
t_2 &= c \\
t_3 &= t' + c - a'_1 + T(a'_1 \geq t').\ \left[a'_1 - t'\right] + T(t' \geq a'_1 + a'_2 + a'_4 + 1).\ \left[c - t' + a'_1 + a'_3\right] \\
t_4 &= 2c + a'_3 \\
t_5 &= t' + 2c + T(a'_3 \geq t').\ \left[a'_3 - t'\right] + T(t' \geq a'_2 + a'_3 + a'_4 + 1).\ \left[c - t' + a'_2 + a'_3\right] \\
t_6 &= 3c + a'_2 + a'_3 \\
t_7 &= t' + 3c - a'_1 + T(a'_1 + a'_2 + a'_3 \geq t').\ \left[a'_1 + a'_2 + a'_3 - t'\right]
\end{aligned} \qquad 2.64$$

It can also be shown that when a function has its input x_1 replaced by x_1 exclusively-ored with k variables, where $k \geq 2$, it too can be converted into a multi-threshold solution, and thus a general solution for k variables, where the initial function has p thresholds, is:

$$\begin{aligned}
a_1 &= c \\
a_i &= a'_i + c \qquad i = 2 \text{ to } k \\
a_i &= a'_i \qquad\quad i = k + 1 \text{ to } n
\end{aligned}$$

$$t_{j+(q-1)(k+1)} = t'_q + (j-1)c - \left[\frac{1+(-1)^j}{2}\right] a'_1 + T\left(t'_q \geq \left[\frac{1+(-1)^j}{2}\right] \cdot a'_1\right.$$

$$+ \sum_{i=k+1}^{n} a'_i + 1 + \sum_{i=2}^{j+1} a'_i - a'_{j+1} - T(j \geq k+1) \cdot \left[a'_j\right]\left[c - t'_q + \left[\frac{1+(-1)^j}{2}\right]\right.$$

$$\left. + \sum_{i=k+1-j}^{k} a'_i - a'_{k+1-j} - T(j \geq k+1) \cdot \left[a'_1\right]\right] + T\left(\left[\frac{1+(-1)^j}{2}\right] \cdot a'_1\right.$$

$$+ \sum_{i=k+1-j}^{k} a'_i - a'_{k+1-j} - a'_{k+2-j} + T(1 \geq j) \cdot \left[a'_{k+1}\right] \geq t'_q) \cdot \left[-t'_q + \left[\frac{1+(-1)^j}{2}\right]\right.$$

$$\left.\left. + \sum_{i=k+1-j}^{k} a'_i - a'_{k+1-j} - a'_{k+2-j} + T(1 \geq j) \cdot \left[a'_{k+1}\right]\right]\right]$$

for $j = 1$ to $k+1$

$q = 1$ to p 2.65

and is p is odd then include the thresholds:

$$t_{p(k+1)+j} = jc + \sum_{i=k+1-j}^{k} a'_i - a'_{k+1-j} \qquad \text{for } j = 1 \text{ to } k$$

where $$c = \sum_{i=k+1}^{n} a'_i + 1 + a'_2 - a'_k$$

As in the case of output spectral translation, this general solution is cumbersome to use for calculating solutions by hand, and therefore the method of finding the thresholds by stepping through the table of $\sum_{i=1}^{n} a_i x_i$ values is suggested here also.

Example

$$f(x) = \overline{x}_1.x_2.\overline{x}_3 + x_1.x_2.x_3 + \overline{x}_1.\overline{x}_2.x_3.x_4$$

The spectrum of this function is:

r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
6	-2	6	2	2	-2	-6	2	2	2	-2	10	2	-2	-2	-2

Comparing the first five coefficients with the entries in the tables of Chow's parameters shows that there is no entry of 66222, and hence the function is not threshold. Input spectral translation of the input x_1 replaced by $x_1 \oplus x_2 \oplus x_3$ results in the spectrum:

r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
6	10	6	2	2	-6	-2	-2	2	2	-2	-2	-2	2	-2	2

There is an entry of 106622 in the tables, so that this function is threshold and the corresponding a parameters are:

a_0	a_1	a_2	a_3	a_4
2	3	2	1	1

which gives a threshold value from equation 2.7 of:

$$t = 5$$

Figure 2.19 shows the realisation of this function.

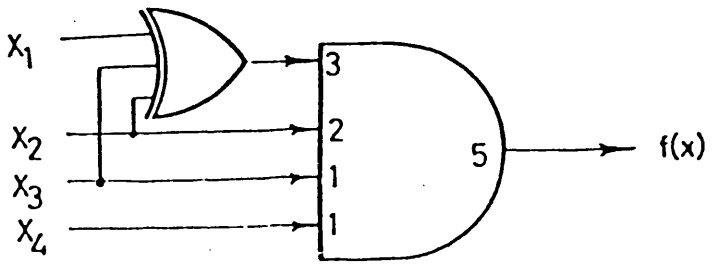


Figure 2.19 Realisation of f(x) using input spectral translations

In this instance the weights are already positive and arranged in proper order, so that the conversion can take place immediately. Using equation 2.65, where k = 3, n = 4:

$$c = \sum_{i=4}^{4} a'_i + 1 + a'_2 - a'_3 = 1 + 1 + 2 - 1 = 3$$

Therefore $a_1 = 3$, $a_2 = 5$, $a_3 = 4$, $a_4 = 1$

The thresholds are:

$k = 3$, $n = 4$, $p = 1$, $t' = 5$

$$t_j = 5 + (j-1)3 - \left[\frac{1+(-1)^j}{2}\right].3 + T(5 \geq \left[\frac{1+(-1)^j}{2}\right].3 + 1 + 1$$

$$+ \sum_{i=2}^{j+1} a'_i - a'_{j+1} + T(j \geq 4).\left[a'_j\right]).\left[3 - 5 + \left[\frac{1+(-1)^j}{2}\right].3 + \sum_{i=4-j}^{3} a'_i - a'_{4-j}\right.$$

$$\left. - T(j \geq 4).\left[a'_1\right]\right] + T(\left[\frac{1+(-1)^j}{2}\right].3 + \sum_{i=4-j}^{3} a'_i - a'_{4-j} - a'_{5-j}$$

$$+ T(1 \geq j)\left[a'_4\right] \geq 5).\left[-5 + \left[\frac{1+(-1)^j}{2}\right].3 + \sum_{i=4-j}^{3} a'_i - a'_{4-j} - a'_{5-j}\right.$$

$$\left. + T(1 \geq j).\left[a'_4\right]\right]$$

for j = 1 to 4

$\therefore$ $t_1 = 5 + T(5 \geq 2).[-2] + T(0 \geq 5).[-5] = 3$

$t_2 = 5 + 3 - 3 + T(5 \geq 3 + 2 + 2).[-2 + 3 + 1] + T(3 \geq 5).[-5 + 3] = 5$

$t_3 = 5 + 6 + T(5 \geq 2 + 2 + 1).[-2 + 2 + 1] + T(1 \geq 5).[-5 + 1] = 12$

$t_4 = 5 + 9 - 3 + T(5 \geq 3 + 2 + 2 + 1).[-2 + 3 + 2 + 1]$

$+ T(3 + 2 + 1 \geq 5).[-5 + 3 + 2 + 1] = 12$

Since p = 1, i.e. odd, include the thresholds:

$$t_{4+j} = 3j + \sum_{i=4-j}^{3} a'_i - a'_{4-j} \qquad \text{for } j = 1 \text{ to } 3$$

$\therefore$ $t_5 = 3$

$t_6 = 6 + 1 = 7$

$t_7 = 9 + 2 + 1 = 12$

Note that $t_1 = t_5 = 3$, and so they cancel out, and that $t_3 = t_4 = t_7 = 12$, so that any two of these cancel out. Thus the final weight-threshold vector is:

3 5 4 1 ; 5, 7, 12

as shown in Figure 2.20.

The alternative method of finding the thresholds by comparing the function f(x) with the values of $\sum_{i=1}^{n} a_i x_i$ is shown in Figure 2.21.

Figure 2.21(a) shows that the function changes from 0 to 1 at values of $\sum_{i=1}^{n} a_i x_i$ of 5 and 12, and changes from 1 to 0 at value of 7, which can be represented as in Figure 2.21(b).

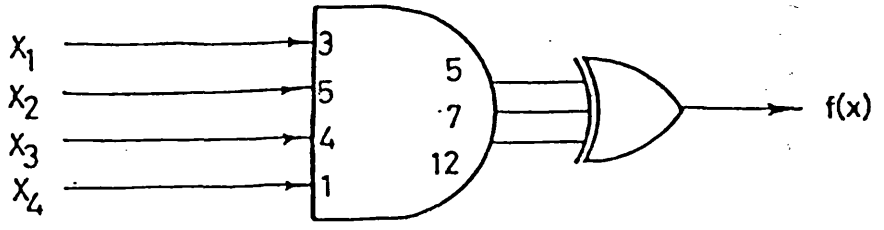


Figure 2.20 Multi-threshold solution to function f(x)

a)

	3	5	4	1		
x_1	x_2	x_3	x_4	$\sum_{i=1}^{n} a_i x_i$	f(x)
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	4	0
0	0	1	1	5	1
0	1	0	0	5	1
0	1	0	1	6	1
0	1	1	0	9	0
0	1	1	1	10	0
1	0	0	0	3	0
1	0	0	1	4	0
1	0	1	0	7	0
1	0	1	1	8	0
1	1	0	0	8	0
1	1	0	1	9	0
1	1	1	0	12	1
1	1	1	1	13	1

b)

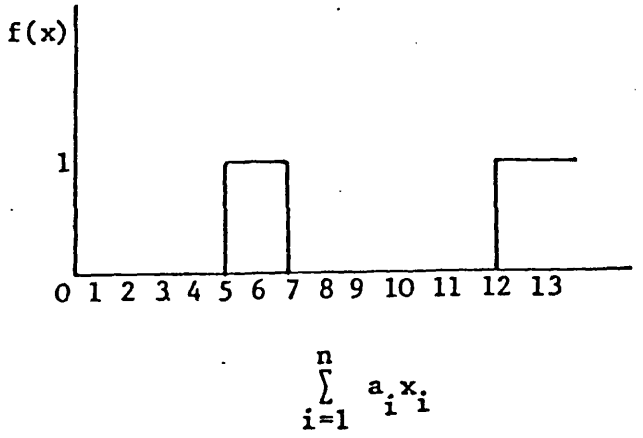


Figure 2.21 (a) Truth table of function f(x)
(b) Graphical illustration of multi-threshold solution

2.5 Conclusions and Further Work

A method for function synthesis has been presented which converts the following into multi-threshold form:

a) functions exclusively-ored with any number of input variables, provided that they can initially be expressed as weight-threshold vectors, using equation 2.45,

b) functions which have one of their input variables replaced by the exclusive-or of that input and any number of other, provided that they can be initially expressed as weight-threshold vectors, using equation 2.65.

Examples have been presented which give an indication of the type of solutions found. As mentioned in section 2.3, tables exist [16,17] which give solutions for $n \leq 4$, and thus a comparison can be made. Listed below are the weight-threshold vectors of the two examples previously shown in Figures 2.16 and 2.20, and the equivalent vectors obtained from the tables.

Example 1

a)	2 -4 4 1 ; 0, 3, 6	Solution developed herewith.
b)	2 3 -3 1 ;-1, 3	Haring and Ohori, and Mow and Fu's previously published solution.

Example 2

a)	3 5 4 1 ; 5, 7, 12	Solution developed herewith.
b)	-3 6 2 1 ; 1, 2, 4, 7	Haring and Ohori's previously published solution
c)	-2 3 2 1 ; 3, 5	Mow and Fu's previously published solution.

Clearly the solutions developed herewith are not optimal, but there is not a great deal of inefficiency. However, this method can be applied to functions of $n > 4$. In general then any function which can be reduced to a threshold function, or a function which has a known multi-threshold equivalent, using spectral translation can be converted to a multi-threshold solution by the application of the two equations 2.45 and 2.65.

An area of further work which has been illustrated as a result of this work is that a function with a particular weight-threshold vector can be converted to another function by adjusting the weights and threshold. From previous publications all that could be done was:

a) invert an input variable x_i by negating the relevant weight a_i and subtracting a_i from the thresholds.
b) invert the whole function by negating all the weights and thresholds and then adding one to the thresholds.
c) permutate any two input variables by exchanging their weights.

Now it has been shown that, for example:

a) the OR function $f'(x) + x_i$ can be obtained by adding c to a_i', where $c = t' - a_i'$,
b) the AND function $f'(x).x_i$ can be obtained by adding c to a_i' and to t', where $c = \sum_{j=1}^{n} a_j + 1 - a_i' - t'$,
c) the exclusive-or function $f'(x) \oplus x_i$ can be obtained by adding c to a_i' and including another threshold of value $t' + c$, where c must be greater than or equal to the two values of c above.

More complex functions too can be created by the process of adding constants c and increasing the number of thresholds. The potential of this procedure has not been pursued, but one consequence is that it may possibly be used to find the weight-threshold vector of a function more directly, say by taking a simple initial function and adding to it, and thus saving a great deal of computational time. However, such a method has not yet been formally researched.

References

1. Dertouzos, M.L.: "Threshold Logic: A Synthesis Approach", (MIT Press, Cambridge, Mass., 1965).

2. Muroga, S.: "Threshold Logic and its Applications", (John Wiley Interscience, New York, 1971).

3. Lewis, P.M., Coates, C.L.: "Threshold Logic", (John Wiley, New York, 1967).

4. Hurst, S.L.: "The Logical Processing of Digital Signals", (Crane-Russak, New York, and Edward Arnold, London, 1978).

5. Chow, C.K.: "On the characterisation of threshold functions", Switch Theory and Logical Design, IEEE Special Publ., 5, 134, pp.34-38, Sept. 1961.

6. Winder, R.D.: "Chow parameters in threshold logic", J. Ass. Computing Mach. 18(2), pp.265-289, April 1971.

7. Winder, R.D.: "Threshold functions through n = 7", Sci. Rep. No.7, AFCRL, Contract AF 19(604)-8423, October 1964.

8. Muroga, S., Tsuboi, T., Baugh, C.R.: "Enumeration of threshold functions of eight variables", Rep. No.245, Univ. of Illinois, August 1967. Abridged in Trans. IEEE, C-19, pp.815-825, Sept. 1970.

9. Hurst, S.L.: "Application of Chow parameters and Rademacher-Walsh matrices in the synthesis of binary functions", Computer J.16, pp.165-173, May 1973.

10. Lloyd, A.M.: "A consideration of orthogonal matrices, other than the Rademacher-Walsh types, for the synthesis of digital networks", Int. J. Electronics, 47, 2, pp.205-212, 1978.

11. Edwards, C.R.: "The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis", Trans. IEEE, C-24, 1, pp.48-62, Jan. 1975.

12. Edwards, C.R.: "Matrix methods in combinational logic design", Ph.D. Thesis, University of Bath, UK, 1973.

13. Edwards, C.R. : "Characterisation of threshold functions under the Walsh transform and linear translation", Electron. Lett., 11, 23, pp.563-565, Nov. 1975.

14. Haring, D.H.: "Multi-threshold elements", Trans.IEEE, EC-15, 1, pp.45-65, Feb. 1966.

15. Hurst, S.L.: "The application of multi-output threshold-logic gates to digital network design", Proc.IEE, 123, 2, pp.128-134, Feb. 1976.

16. Haring, D.H., Ohori, D.: "A tabular method for the synthesis of multi-threshold threshold elements", Trans.IEEE, EC-16, 4, 216-20, April 1967.

17. Mow, C.-W., Fu, K.-S.: "An approach for the realisation of multi-threshold threshold elements", Trans.IEEE, C-17, 1, pp.32-46, Jan. 1968.

18. Ercoli, P., Mercurio, L.: "Threshold logic with more than one threshold", Proc. 1962 IFIPS Congress, Amsterdam, North Holland, pp.741-745, 1962.

19. Picton, P.D.: "Realisation of multi-threshold threshold logic networks using the Rademacher-Walsh transform", Prof. IEE, Pt.E, 128, 3, pp.107-113, May 1981.

CHAPTER 3

CHARGE-COUPLED DEVICES (C.C.D.'s)

3. Charge-Coupled Devices (C.C.D.'s)

3.1 General Description [2,3,4,5]

3.1.1 Creation of a Well

The C.C.D. was invented in 1969 by Boyle and Smith [1]. However, it was not really a new device but an alternative way of using an older and more familiar device, the Metal-Oxide-Semiconductor (M.O.S.) capacitor. Figure 3.1 shows a cross section of an n-channel M.O.S. capacitor consisting of a p-type silicon layer or substrate, a silicon dioxide layer, and a metal plate known as the gate. A p-channel device would be identical except for an n-type silicon substrate and in the following discussion all voltage polarities would be reversed.

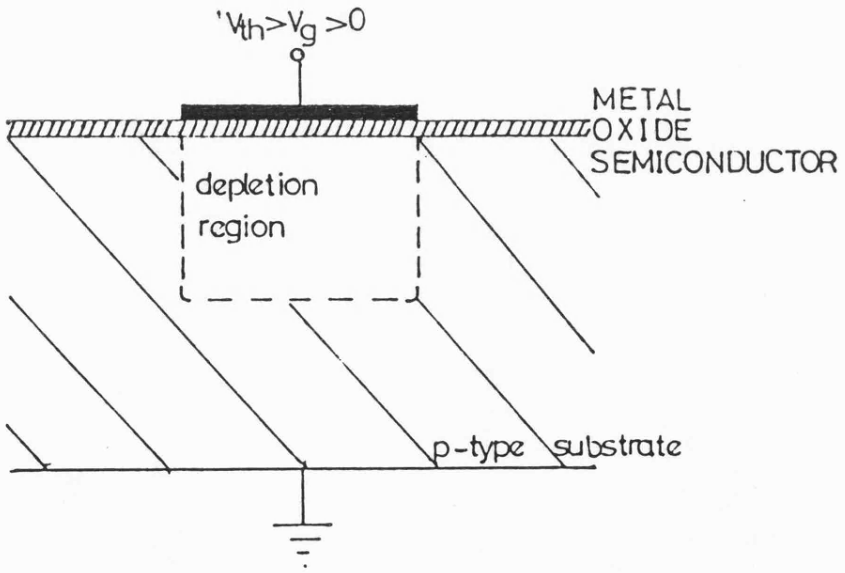


Figure 3.1 M.O.S. capacitor

The p-type substrate is doped with acceptor ions [24] which can be considered as "free holes" or mobile positive charges. When a positive voltage is applied to the gate, therefore, the holes are repelled forming a depletion region immediately under the gate and extending into the substrate. Increasing the voltage increases the extent of the depletion region until a critical point is reached when the applied voltage V_g equals a threshold voltage V_{th}. Then, if electrons are available, an inversion layer or very thin n-channel forms at the semiconductor-oxide interface. In the case of an M.O.S. transistor, electrons are available from the source or drain, but in the M.O.S. capacitor their only origin is from the thermally (or optically) generated electron-hole pairs which occur at finite intervals of time. Therefore, if the applied voltage is suddenly pulsed to a voltage beyond the threshold voltage the depletion region will extend initially far into the substrate; a situation known as deep depletion. As time passes, electrons will be generated and will start to form the inverse layer and the depletion region will shorten. Ultimately the state when the device is said to be in equilibrium is reached, where the depletion region is the same depth as at the onset of inversion. Figures 3.2 and 3.3 show the relationship between the gate voltage V_g, the voltage at the semi-conductor-oxide interface known as the surface potential ϕ_s, and the charge in the inversion layer Q_{inv}.

The diagrams indicate that the threshold voltage V_{th} is defined as the gate voltage at which the surface potential ϕ_s equals twice the Fermi potential, ϕ_F, which in this instance is approximately 0.6V. The reason for this would require a fuller explanation which is not necessary for this thesis but which can be found in any semiconductor physics textbook [24]. Above the threshold value, however, it can be

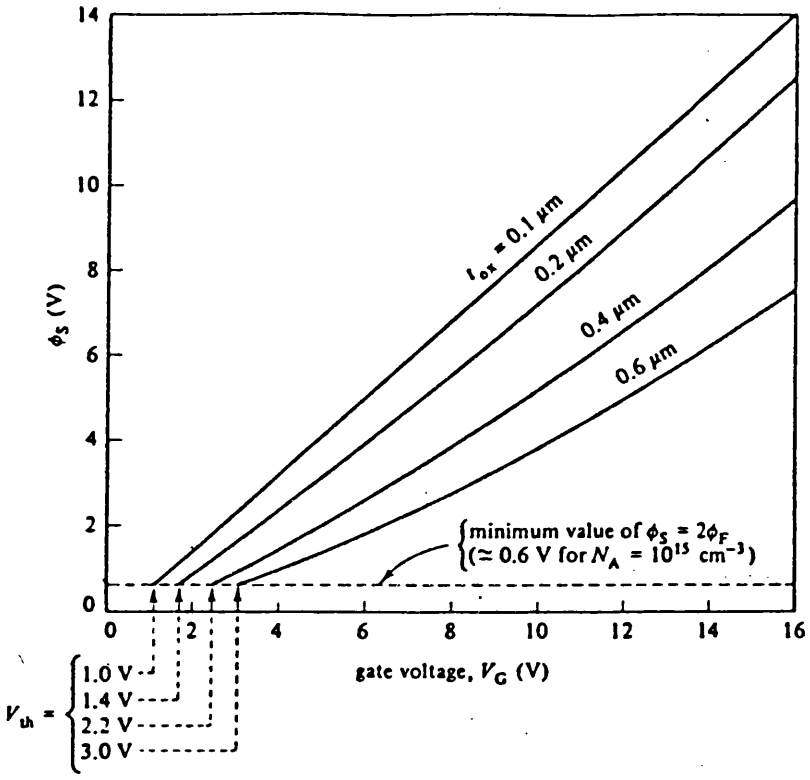


*Figure 3.2 Variation of ϕ_s with V_G for M.O.S. structures of different oxide thickness t_{ox} fabricated on a p-type substate with doping level 10^{15} cm^{-3}, with inversion charge = 0

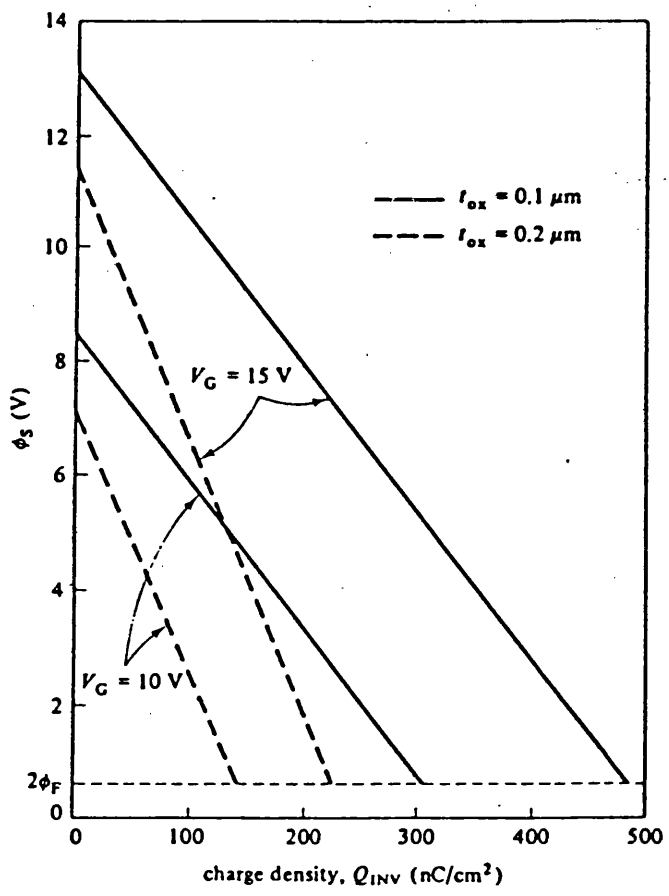


*Figure 3.3 Variation of ϕ_s with Q_{inv} for two of the M.O.S. structures in Figure 3.2, when V_G = 10V and 15V

* Reproduced from Beynon and Lamb [5]

seen that there is a very good linear relationship between the quantities V_g, ϕ_s, and Q_{inv}, which is approximated by equation 3.1.

$$\phi_s = V_g + \frac{Q_{inv}}{C_{ox}} \qquad 3.1$$

where C_{ox} is the oxide capacitance per unit area.

This linear relationship can be used to form a simple model where the region under the gate is regarded as a well, the charge is analogous to a liquid in the well, and the surface potential is regarded as the distance from the top of the well to the surface of the liquid as in Figure 3.4.

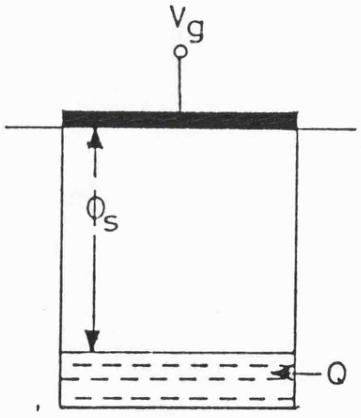


Figure 3.4 Model of an M.O.S. capacitor

Note that this is only a schematic diagram, so that the well must not be confused with the depletion region. The maximum amount of charge that can be stored in a well, known as its "depth" or charge handling capacity, can be found by letting ϕ_s equal zero in equation 3.1, and

multiplying Q_{inv} by the area of the electrode, as in equation 3.2.

$$Q_{max} = - C_{ox} . AV_g \qquad 3.2$$

where A is the area of the electrode.

A gate like this would generally be in a state of non-equilibrium, since the charge under it would rarely be required to be equal to Q_{max}. Therefore an amount of charge stored in this way would have to be repeatedly refreshed so as to avoid degradation by the electron-hole pair generation. Alternatively, the charge could be transferred to other wells fast enough to avoid corruption.

3.1.2 Charge Transfer

A typical cross section of a C.C.D. is given in Figure 3.5. It consists of a number of gates placed serially between the input diode and gate and the output diode and gate. Each gate can have a voltage applied to it to create a well with a charge handling capacity as in equation 3.2.

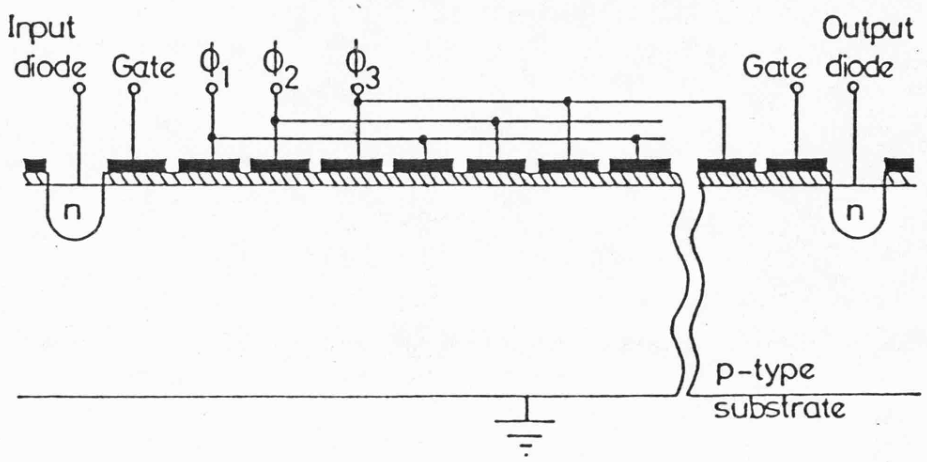


Figure 3.5 Typical cross-section of a C.C.D.

Generally the input diode is held at a high positive voltage and is therefore reverse biased. However, if it is pulsed to a low voltage, charge will overflow via the input gate to the well under ϕ_1, as in Figure 3.6. This charge can then be transferred to the well under ϕ_2 as in Figure 3.7.

Figure 3.7(a) shows a well which contains some charge. In Figure 3.7(b) an adjacent well has a voltage V suddenly applied to it, which, if the gates are close enough together, causes the two wells to merge and the charge to distribute itself equally between the two. Figure 3.7(c) shows the first well having its gate voltage slowly decreased so that its remaining charge flows into the adjacent well until finally the second well contains all the charge. Clearly this process could be continued to a third well, thus giving the facility of charge transfer. A three phase clocking system which allows this is shown in Figure 3.8.

Other clocking schemes have also been presented which only require two phases and even a one phase system [3,5]. However, the three phase is the most commonly encountered since the others require more complicated fabrication processes.

Finally, the charge reaches the output where it overflows the output gate into the output diode which normally has a positive voltage applied to it. Typically a capacitor and amplifier is placed at the output to detect this charge and give a voltage proportional to it.

It is the C.C.D.'s ability to store charge and to transfer it which has resulted in its three main applications, namely, image sensing, analogue signal processing, and digital memories. Recently, interest has been shown in developing C.C.D. logic gates for use in digital signal

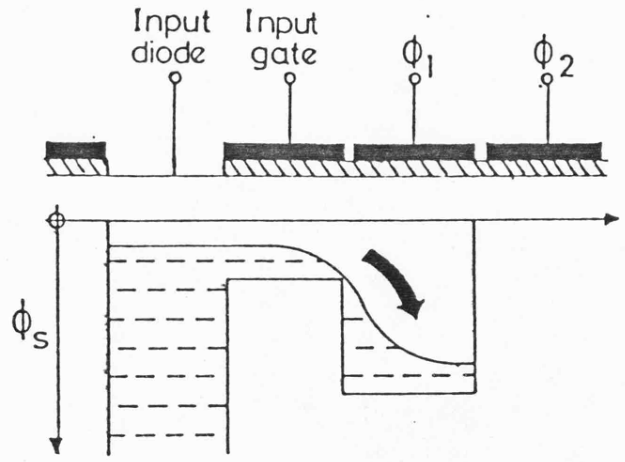


Figure 3.6 Charge input

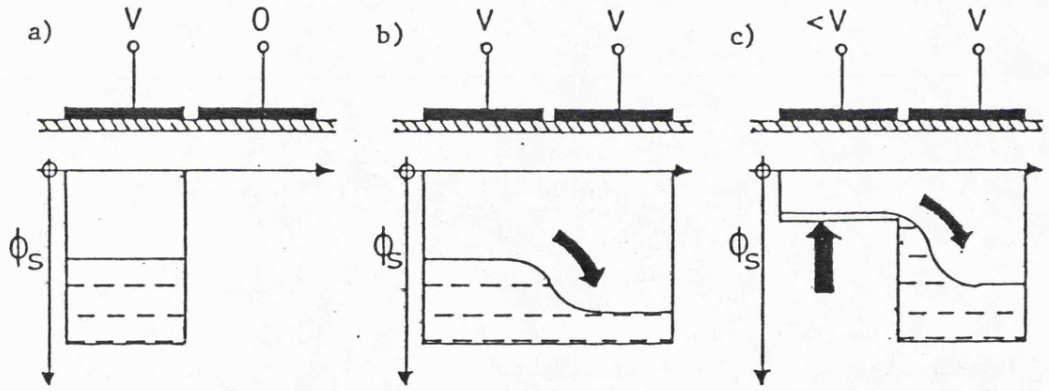


Figure 3.7 Charge transfer

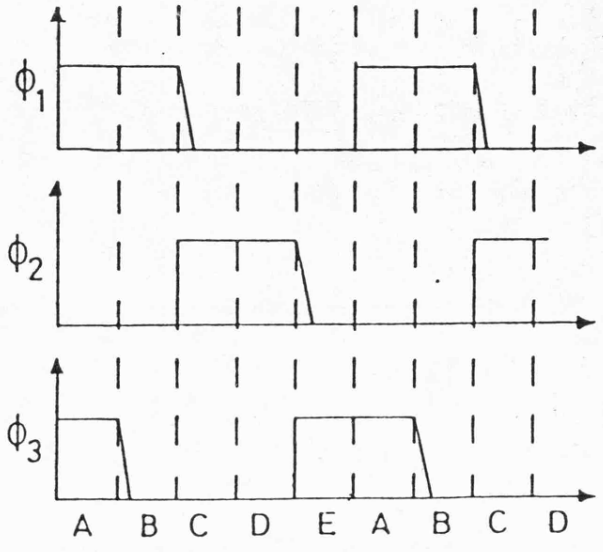


Figure 3.8 Three phase clocking system

processing, which opens up a completely new area of application.

3.2 Logic Design

As mentioned already the C.C.D. has received a great deal of attention in the area of digital memories, where the presence or absence of a unit charge packet corresponds to a stored logic 1 or logic 0 respectively. Clearly this idea can be extended to digital signal processing using properties of the C.C.D. to perform simple logic functions [7]. This means that there is the possibility of combining dense memory arrays and complex digital signal processors on the same chip. So far this idea has been approached in two ways; firstly, simple binary logic functions and secondly, multi-valued logic, mainly four-valued (quaternary).

3.2.1 Some Additional C.C.D. Properties

In order to follow the operation of C.C.D. logic circuits it is necessary to understand the following three functions:

a) charge summing,

b) charge overflow,

c) charge sensing.

3.2.1.1 Charge Summing

Earlier it was discussed how a C.C.D. stores charge in a well and how this charge can be transferred to another well using a three phase clocking system. Consider what happens when charge from two wells are clocked into the same well. Schematically this is shown in Figure 3.9.

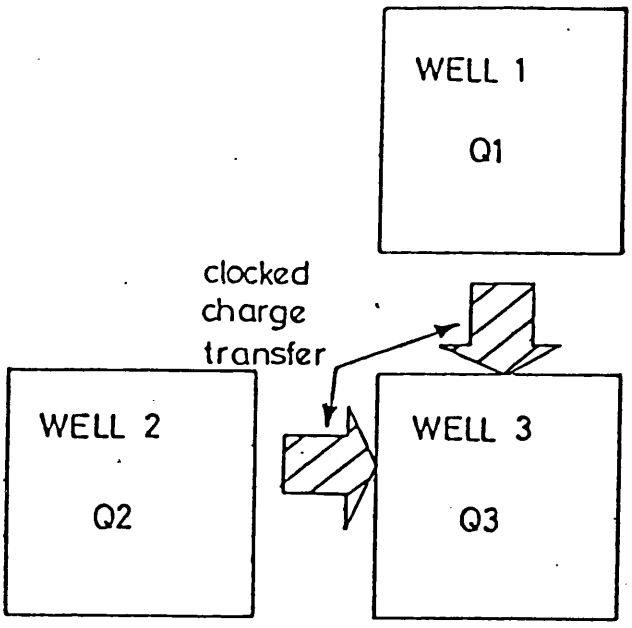


Figure 3.9 Charge summing

Clearly, if well 3 has a large enough charge handling capacity, then Q_3 will be the sum of Q_1 and Q_2, as in equation 3.3.

$$Q_3 = Q_1 + Q_2 \qquad 3.3$$

More than two wells can obviously be used, provided that the summing well has a large enough charge handling capacity.

3.2.1.2 Charge Overflow

If a well contains charge Q_1, and this charge is then clock transferred to a second well whose charge handling capacity is less than Q_1, then the excess charge can be made to overflow into a third well via a barrier gate, shown schematically in Figure 3.10.

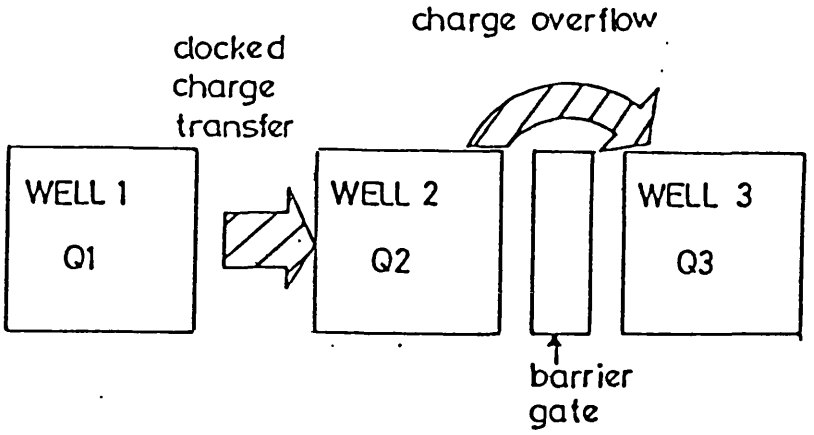


Figure 3.10 Charge overflow

If the third well has a large enough charge handling capacity to overflowing charge, then Q_1, Q_2 and Q_3 are related by equation 3.4.

$$Q_1 - Q_2 = Q_3 \qquad 3.4$$

The barrier gate is simply an electrode with a low voltage applied, similar to the input gate shown in Figure 3.6. Another barrier gate and well could be placed after well 3 so that if the charge handling capacity of well 3 is less than the charge that it receives, a further overflow could occur into the fourth well, and possibly into a fifth, sixth, etc.

3.2.1.3 Charge Sensing

Although charge can be detected destructively at the output as described earlier, it can also be sensed at any point without altering it whatsoever. A charge sensing amplifier detects the presence of charge under an electrode and alters the voltage on another electrode by an amount proportional to the sensed charge. A typical structure is the master-slave floating gate shown in Figure 3.11 [8].

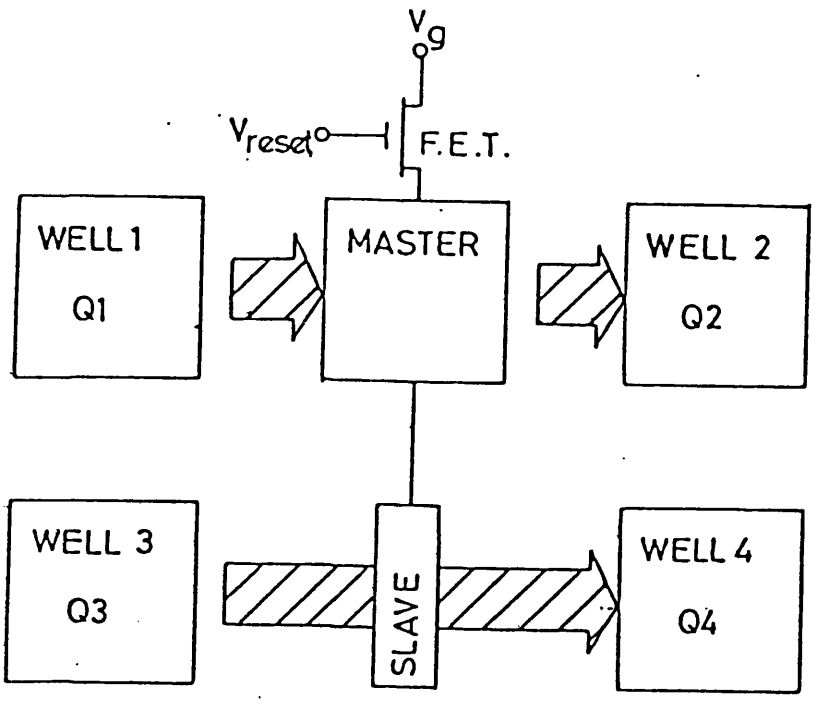


Figure 3.11 Master-slave floating gate

The voltage on the master and the slave wells is set by the F.E.T. and kept constant by the parasitic capacitances. Charge entering the master well by a clocked transfer from well 1 will cause a drop in the voltage on the master electrode and so a corresponding drop must also occur on the slave electrode. If the drop is sufficient the slave will block all transfer of charge from well 3 to well 4. After the charge has been transferred from the master well, the master and slave are reset to their original voltages. Thus the function of the structure can be given by equation 3.5.

$$Q_4 = Q_3 \quad \text{if} \quad Q_1 < Q_{th}$$
$$Q_4 = 0 \quad \text{if} \quad Q_1 > Q_{th} \qquad 3.5$$

where Q_{th} is some arbitrary threshold charge.

3.2.2 Binary Logic

Binary logic uses unit charge packets, the presence or absence of which constitutes a logic 1 or 0 respectively. Developments in this area have been mainly concentrated on logic arrays using the very simple AND/OR structure [8,9,10] shown in Figure 3.12.

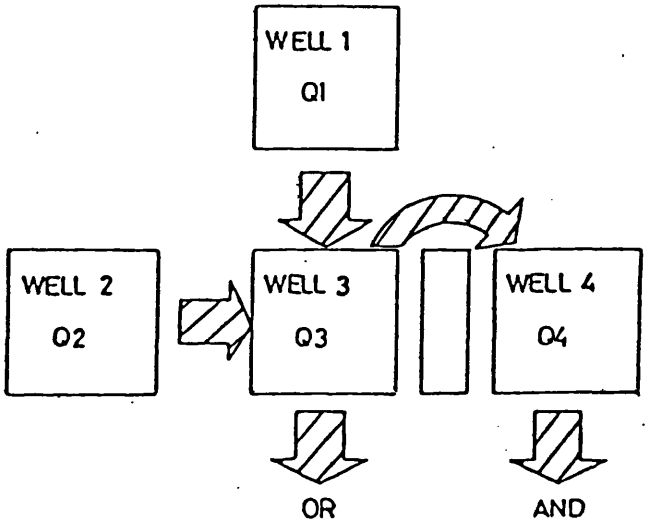
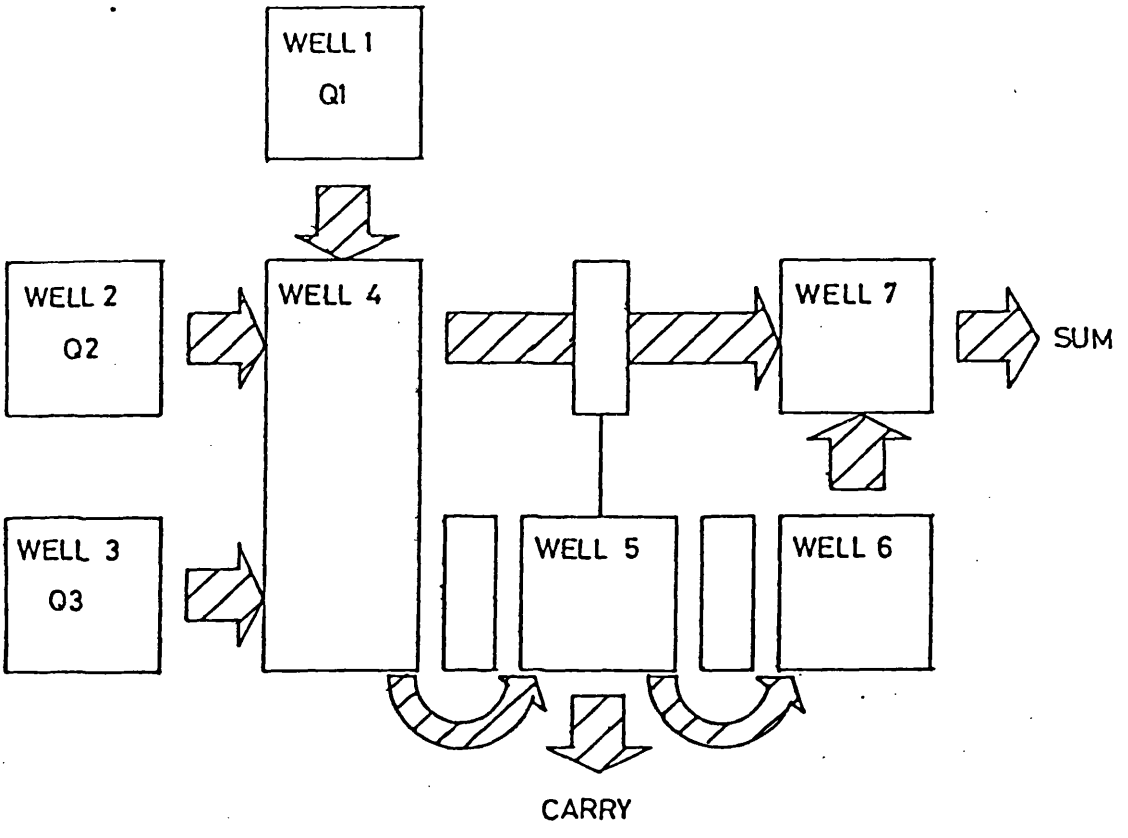


Figure 3.12 AND/OR logic structure

All wells have charge handling capacities of unity. Thus if both Q_1 and Q_2 are unit charge packets, when they are clocked to well 3 there will be a charge overflow of one unit into well 4. Thus there is charge in well 3 if there is charge in well 1 OR well 2, and charge in well 4 if there is charge in well 1 AND well 2. This kind of structure is ideally suited for implementing functions in a pipelined or serial input manner, more of which is discussed in the following chapter. However, arithmetic functions can also be implemented using a slightly modified structure. Figure 3.13 shows a full adder circuit and the corresponding truth table for each of the wells [9].

As in the AND/OR circuit, each well has a charge handling capacity of unity. Thus when two of the charges Q_1, Q_2 or Q_3 enter well 4 there

a)



b)

WELL	1	2	3	4	5	6	7
	0	0	0	0	0	0	0
	0	0	1	1	0	0	1
	0	1	0	1	0	0	1
	0	1	1	1	1	0	0
	1	0	0	1	0	0	1
	1	0	1	1	1	0	0
	1	1	0	1	1	0	0
	1	1	1	1	1	1	1

Figure 3.13 Full adder circuit and truth table

is an overflow into well 5 which then blocks the transfer of charge from well 4 to well 7. However, when all three charges are present charge also overflows into well 6 which is then clocked to well 7, the sum output.

Arrays of these structures can then be used to perform addition, and with extra delays multiplication.

3.2.3 Multi-Valued Logic

C.C.D.'s have been mainly used as either analogue or binary devices. However, they can be used as multi-level devices as in Figure 3.14, where four different amounts of charge can be stored in a well.

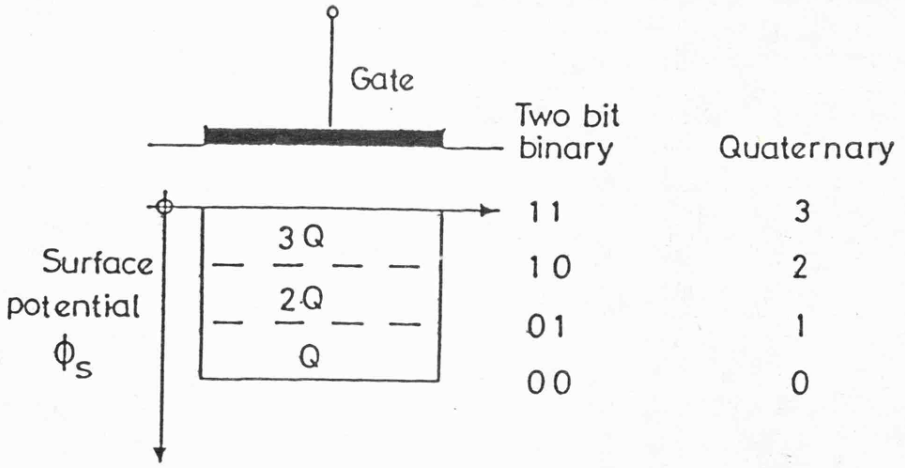


Figure 3.14 Multi-level storage of charge

These charge levels can then be used either to double the storage density of a memory device by considering each level to be the equivalent to two binary bits [11], or as logic levels in quaternary logic [12,13,14]. The main drawback of using multi-levels is the fact

that it is extremely difficult to detect the different levels [2]. The use of four levels has been researched and actual devices made, but for more than four levels very little work has been undertaken, although up to thirty two levels has been reported [15].

As an example of a quaternary device [13] consider a full adder circuit which is identical to that of Figure 3.13(a) except that all the wells now have charge handling capacities of three apart from well 5 which has a capacity of one. Thus the contents of wells 4, 5, 6, and 7 can now be shown in Figure 3.15.

Decimal sum of $Q_1 + Q_2 + Q_3$	well 4	5	6	7
0	0	0	0	0
1	1	0	0	1
2	2	0	0	2
3	3	0	0	3
4	3	1	0	0
5	3	1	1	1
6	3	1	2	2
7	3	1	3	3

Figure 3.15 Contents of the wells of a quaternary full adder

The largest value of the sum is seven because one of the inputs, Q_1 say, is a carry signal from a previous full adder and therefore has a value of 1 or 0.

In fact for any m-valued logic, a full adder circuit would be similar to that of Figure 3.13(a) with all wells having a charge handling capacity of m - 1 except for well 5 which has a charge handling capacity of one.

3.3 Threshold Logic

In the previous section it was shown that charge can be summed, made to overflow, and that multi-level logic is possible. These facts can be shown to be ideal for the production of a threshold logic gate. Reconsider the threshold logic equation:

$$f(x) = 1 \quad \text{iff} \quad \sum_{i=1}^{n} a_i x_i \geq t$$

$$f(x) = 0 \quad \text{otherwise} \qquad 3.6$$

Therefore each input variable x_i can have a charge packet associated with it proportional to the weight a_i, and since a_i is an integer this charge packet can be envisaged as a_i times a basic charge packet, Q say, as in Figure 3.14. Each packet of charge can then be summed into a common well as in equation 3.3, where the charge handling capacity of this well is proportional to t - 1, so that charge of t or more entering it would overflow into another well as in equation 3.4. Thus the charge in this third well would be either equal to zero, or equal to $\sum_{i=1}^{n} a_i x_i - t + 1$. The presence or absence of charge in this well results in an output of a logic 1 or 0 respectively, regardless of the amount of charge.

It is interesting to note that the full adder circuit of Figure 3.13, and particularly the quaternary full adder, the contents of which are shown in Figure 3.15, acts in a very similar manner to a threshold logic gate with three inputs. One of the inputs has a weight of one, and the other two can have weights of up to three. The charges are summed and made to overflow into wells 5 and 6, which can therefore be considered as acting as though they had thresholds of four and five.

The following sections consider how a charge packet proportional to a_i can be formed and how the final charge can be detected at the output.

3.3.1 Charge Input

A method which provides the input of a fixed amount of charge is the charge equilibration, or "fill-and-spill" [2,3,4,5], as shown in Figure 3.16. Initially the input diode is strongly reverse biased and no charge resides under gates 1, 2, or 3 as in Figure 3.16(a). Figure 3.16(b) shows the voltage on the input diode suddenly pulsed to a low voltage so that charge overflows the wells, in particular filling the well under V_3. Finally Figure 3.16(c) shows that the voltage on the diode is again set to its original reverse bias so that the charge spills back leaving only that charge in well 3, the amount of which is given, using equation 3.1, as:

$$Q = -(V_3 - V_2).C_{ox}.A \qquad 3.7$$

The two gates with V_1 and V_2 attached are included instead of the usual one so that the input variable x_i can replace V_1. Thus the previous description occurs when $x_i = 1$, and therefore $V_1 > V_2$. However, if $x_i = 0$, then $V_1 = 0$ and charge from the input diode is blocked as in Figure 3.17.

3.3.2 Charge Detection

As previously noted, the output has to detect the presence or absence of charge regardless of the amount of charge, and hence practical arguments against the use of multi-levels, namely the difficulty in detecting the different levels, do not apply.

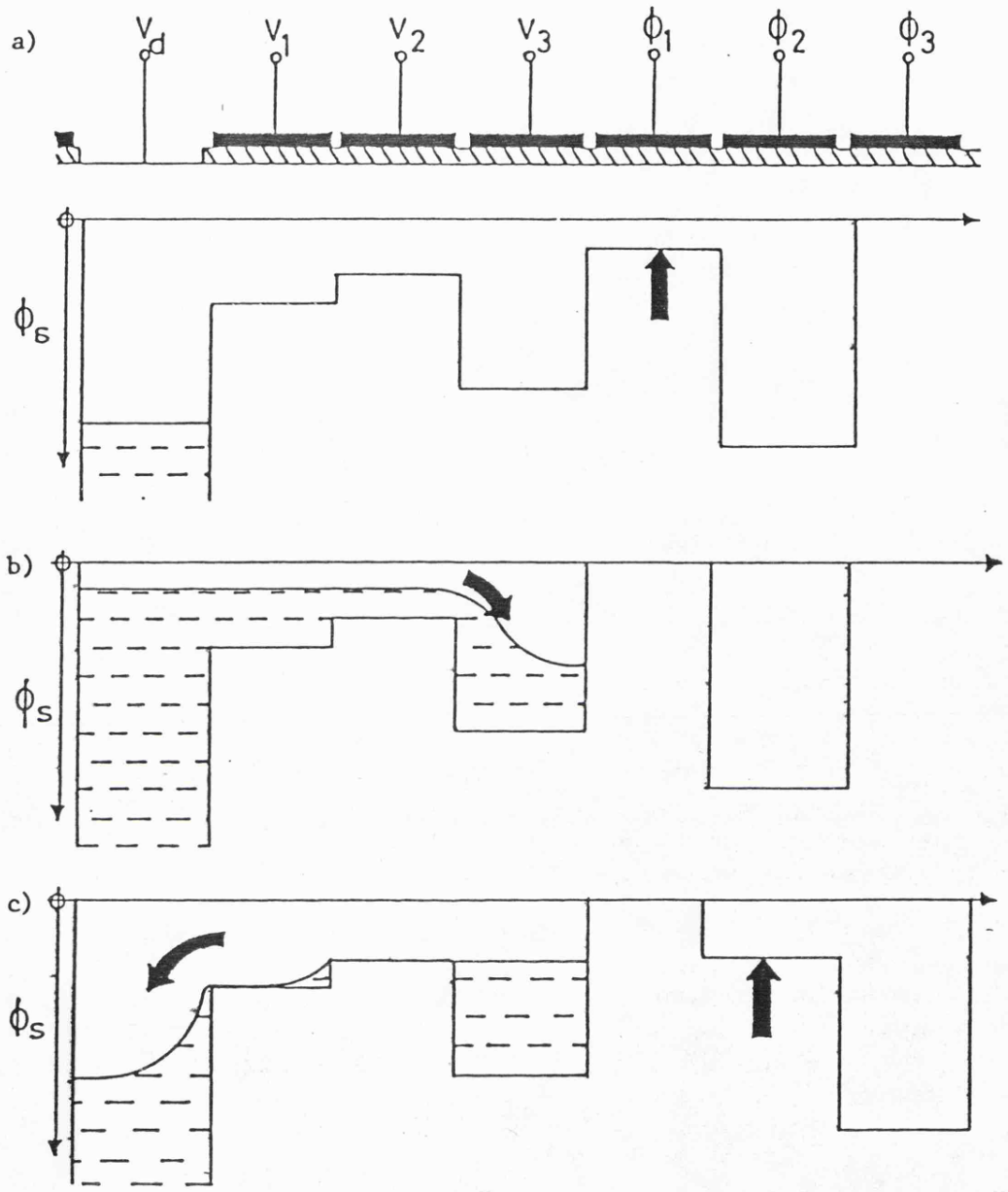


Figure 3.16 Charge equilibration

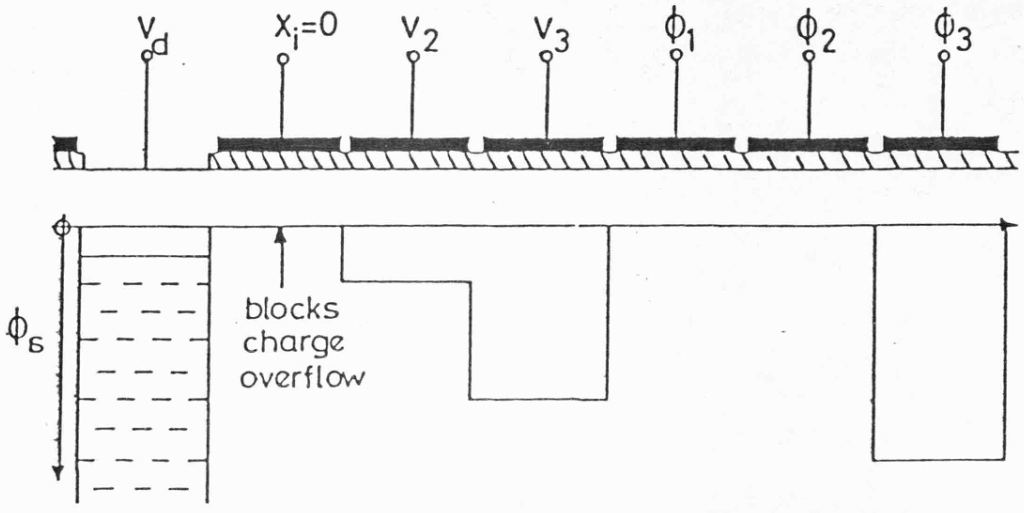


Figure 3.17 Charge blocking when $x_i = 0$

A typical method of detection is shown schematically in Figure 3.18.

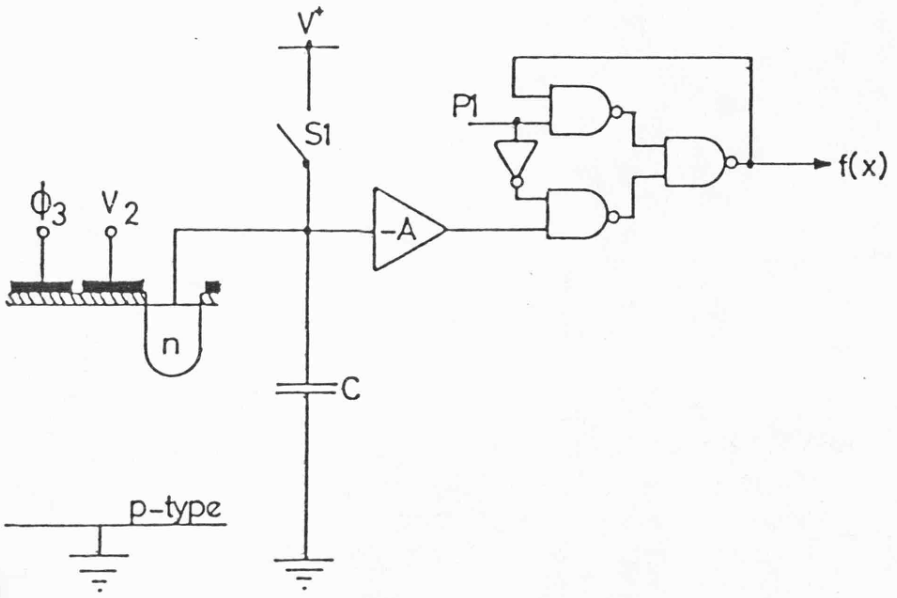


Figure 3.18 Output circuitry

Switch S1 is closed long enough for the voltage on the capacitor to settle at V^+ which then strongly reverse biases the output diode. The switch is then opened and, assuming that the amplifier has a very high input resistance, the voltage on the capacitor remains constant. Any charge under the last clocked electrode will overflow the output gate with the low voltage V_2 on it when the clock phase ϕ_3 drops to zero volts, and will then pass via the output diode to the capacitor, thus lowering the potential across the capacitor. This potential drop is amplified and inverted so that the NAND gate receives it as a logic 1 signal. (If there had been no charge and therefore no voltage drop on the capacitor, the NAND gate would receive a logic 0 signal). The pulse P1 is set to a low voltage, logic 0, so that the output f(x) is equivalent to the logic input from the amplifier. The pulse P1 is then reset to a high voltage, logic 1, which then holds the output f(x) at

the same level even if the signal from the amplifier is altered. The switch S1 is then closed again to discharge the capacitor and the process repeated. All the devices shown, namely switches, amplifier, sample and hold circuit, can be manufactured using M.O.S.F.E.T.'s which are compatible with the C.C.D. and the whole can therefore be incorporated on to the same chip [16].

3.3.3 System Integration

The inputs and outputs described so far have assumed that the threshold logic gate is an independent device receiving logic signals or voltage levels and yielding the same signals or voltages. However, it has been said that the desirable feature of logic circuits in C.C.D.'s is that they can be incorporated into a complete digital processing system, i.e. all C.C.D. devices on one chip. In this case the inputs and outputs that the internal threshold logic gates would obtain would be unit charge packets as in the case of the binary logic gates described earlier. This is easily adopted, however, using the charge sensing amplifier described in section 3.2.1.3, details of which are considered in the following sections.

3.3.3.1 Charge Input

At the input, charge is sensed in a master well and the slave is equivalent to V_2 in Figure 3.16; hence a charge packet of $(V_3 - V_2).C_{ox}A$ is introduced to the threshold logic gate. If no charge is present in the master well then the voltage V_2 is at its maximum, which is set so that $V_1 \geq V_{2max} \geq V_3$, as in Figure 3.19.

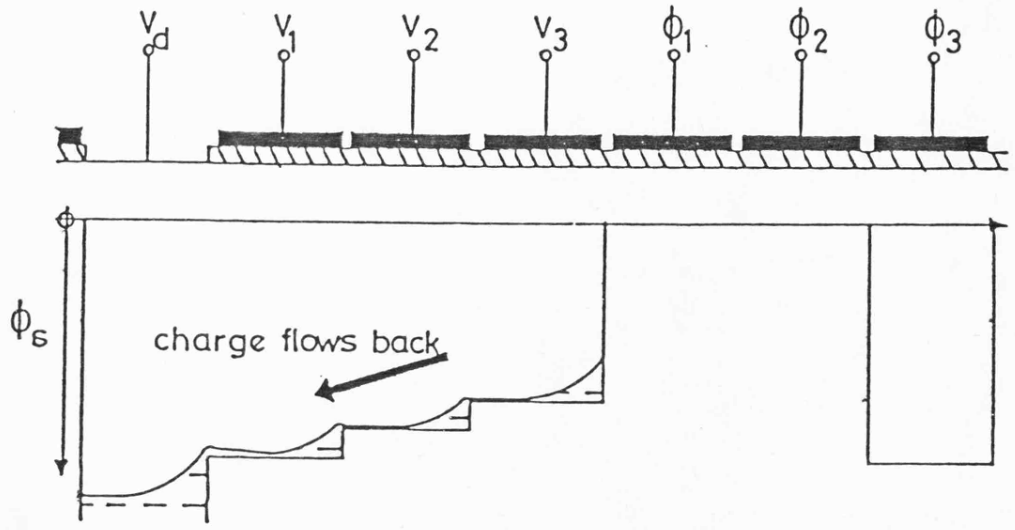


Figure 3.19 Charge blocking

When the input diode is now pulsed all charge will spill back out so that no charge packet is formed.

3.3.3.2 Charge Output

In the description of a threshold logic gate so far, it has been assumed that there is only one output. The previous chapter showed the desirability of having multi-thresholds, and fortunately this is easily incorporated into the C.C.D. by extending the overflow of charge to more than one well, each of which has a charge handling capacity proportional to $t_i - t_{i-1}$. The logical values of each of these wells then have to be exclusively-ored to give the final output. If these logic values are voltage levels from the output of the circuits shown in Figure 3.18 then the exclusive-or gate can also be manufactured using M.O.S.F.E.T.'s [17]. However, if the gate is part of a digital processor, it is more likely that the output will be required to be in the form of a unit charge packet.

If the multi-threshold wells described above are considered as the master wells of the master slave charge sensors, then the slave end could be equivalent to V_1 in Figure 3.16. Thus, if charge is present in the threshold logic gate wells then no charge packet is created in another part of the system, and if there is no charge then a charge packet is introduced. Each of these charge packets can then be transferred to an array of C.C.D. binary exclusive-or gates [9], one of which is shown in Figure 3.20.

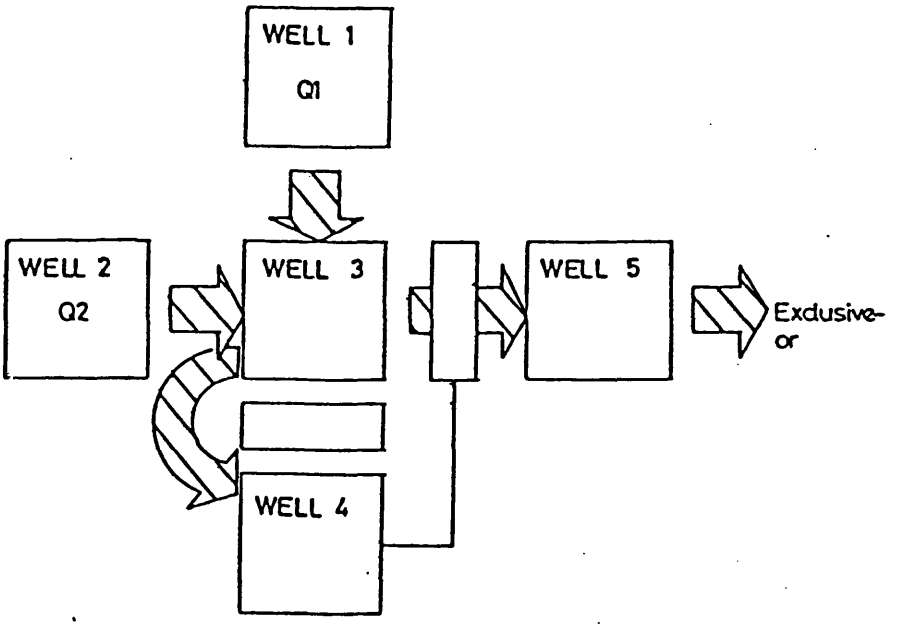


Figure 3.20 Binary exclusive-or gate

The structure of the exclusive-or gate is similar to the AND/OR gate of Figure 3.12 except that well 4 is now a master well of a master-slave charge sensor. Thus if well 4 contains charge, the transfer of charge from well 3 to well 5 is blocked. The charge packets Q_1 and Q_2 are in fact the inverse of what they should be since they are equivalent to a logic 1 if charge is not present in the threshold well and a logic 0 if there is, which is the reverse of what is normally assumed.

However, it makes no logical difference to the final output, since the exclusive-or of two variables is equivalent to the exclusive-or of their inverse.

The previous chapter showed that the "half exclusive-or gate" is all that is necessary in a multi-threshold logic gate. Thus the simpler circuit if Figure 3.21 can be used instead of the exclusive-or gate.

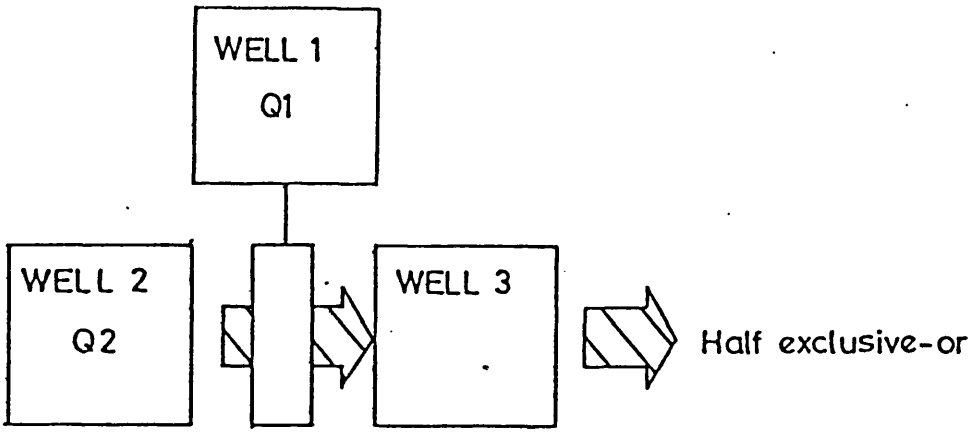


Figure 3.21 Half exclusive-or gate

In this situation, Q_1 can never be a 1 when Q_2 is a 0, so that the output is equal to Q_2 when Q_1 is 0, and equal to 0 when Q_1 is 1. For more than two thresholds, arrays of these gates would be used. Note, however, that the exclusive-or or half exclusive-or of an odd number of variables is inverted if the variables are inverted. Thus if there are an odd number of thresholds and outputs then the last half exclusive-or gate must have Q_2 equal to a logic 1 to invert the final output. This also applies when there is only one threshold, since this too is an odd number. The array of gates would be structured so that the contents of well 3 in each gate run into a common well which would then

contain the output logic value.

3.4 Threshold Logic Gate

Figure 3.22 shows the full structure of the proposed threshold logic gate [19]. It has four inputs and five thresholds and outputs which are a sufficient number to realise any four input variable function.

The gate operates as follows: charge is formed in wells G1 to G4 using the charge equilibration method discussed earlier. The amount of charge is proportional to a_i if x_i is a logic 1, or nothing at all if x_i is a logic 0. It is then transferred using a three phase clock to the large well G5 where it is summed. The charge handling capacity of well G5 is determined by a voltage proportional to $t_1 - 1$, but unlike the voltages on the other wells, this one has to be scaled down by a factor k, where k is the ratio of the area of well G5 to the other wells, which are assumed to be all the same size. Any charge in excess of t_1 overflows into well G6, more still into G7 and so on. Any charge that is present in wells G5 to G10 is then transferred using a three phase clock, this time with the phases reordered to minimise delay and to avoid any backward flow of charge. The charge from G5 goes to a drain for clearing, whereas charge from the other wells go to drains where it is then detected using circuits as described earlier such as in Figure 3.18.

If the circuit is to be incorporated into a system where it will receive binary inputs in the form of unit charge packets and is required to yield charge packets then obviously the gate would be slightly different although its basic structure would remain the same. At the input the two input gates labelled x_i's and V_1 would be interchanged to

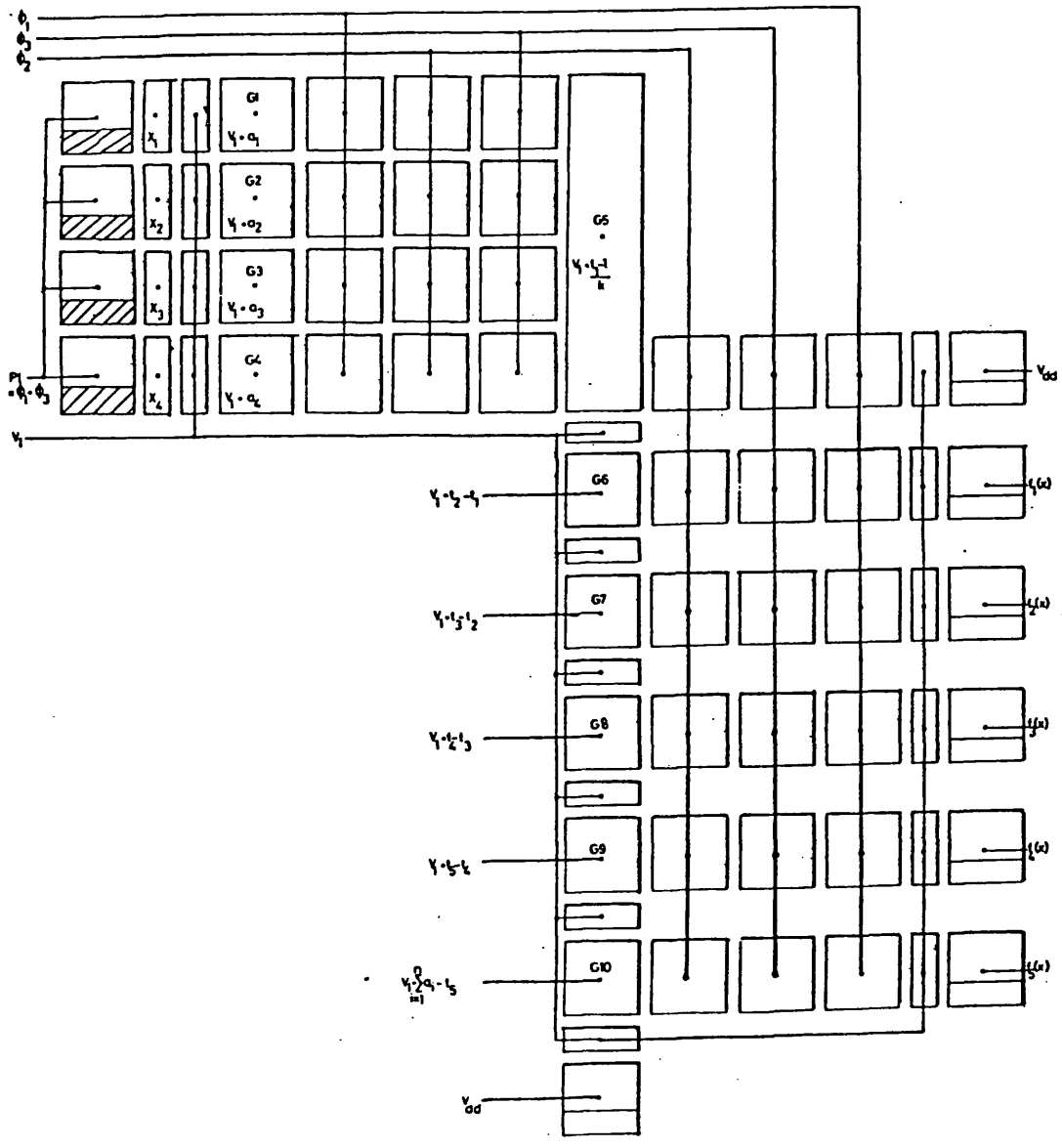


Figure 3.22 C.C.D. multi-threshold logic gate

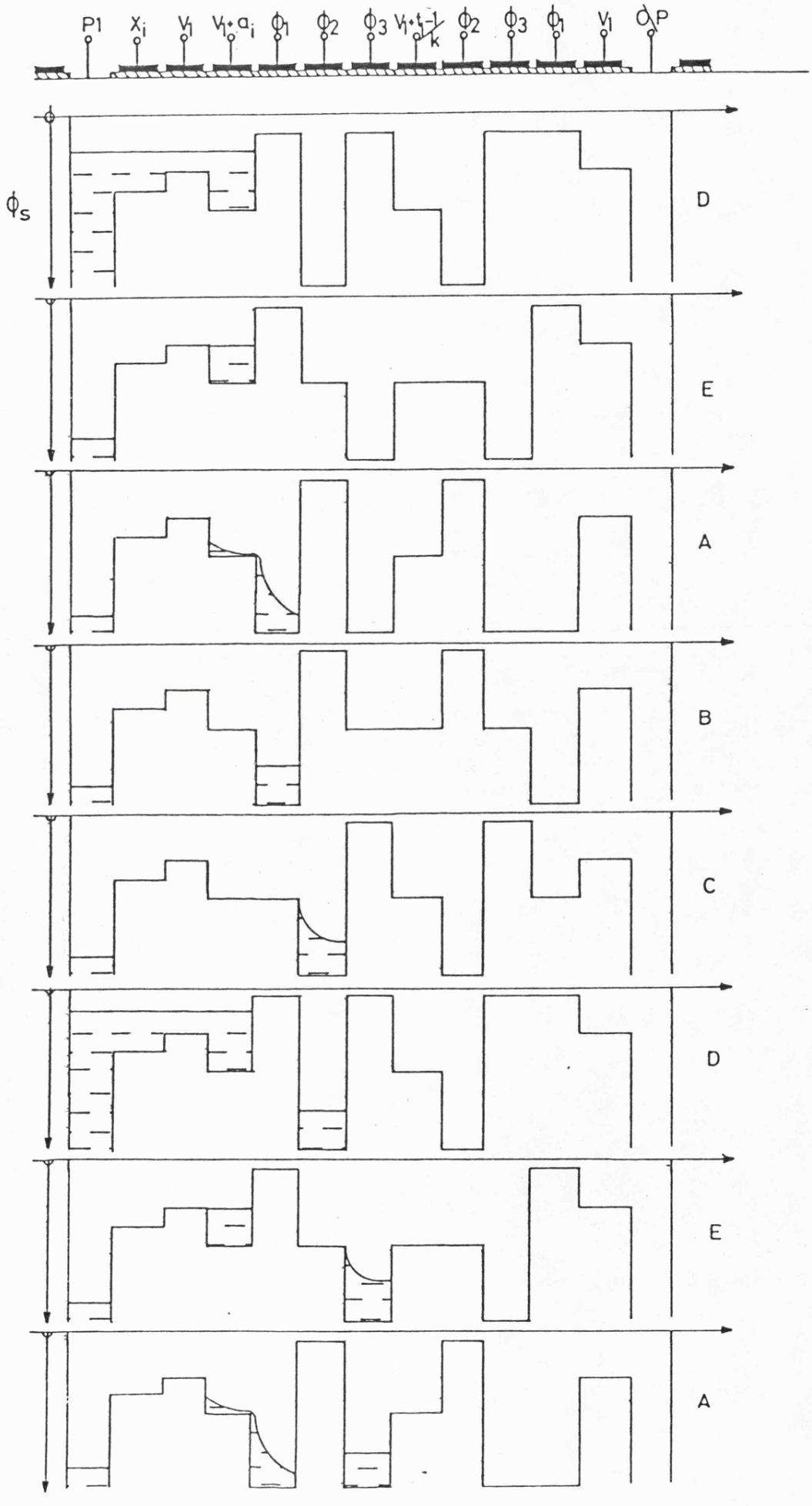
give a situation similar to that of Figure 3.19, and at the output the charge would still be transferred to drains only in this case for all of it to be cleared. This is because wells G6 to G10 would be master wells of charge sensing amplifiers as described in section 3.3.3.2, and so the outputs would already be determined before the charge is transferred to the drains.

Figure 3.23 is a cross section through the device showing the contents of various wells at various stages of operation labelled in accordance with the clock phases shown in Figure 3.8. The cross section is taken through the input line corresponding to x_4, one end of G5, and the output line which in fact clears G5, but this does not alter its generality since the other lines are identical. However, not shown is the process of charge overflow into wells G6 to G10 since this occurs effectively out of the plane of the paper.

It can be seen from this figure that there is latency inherent to the system, since the inputs are sampled at intervals of one clock period, and the delay between the relevant output being formed is three clock periods.

3.4.1 Limitations of the Threshold Logic Device

One of the most serious limitations of all C.C.D.'s is that at each charge transfer, a small fraction of charge is left behind. This fraction $\epsilon \left[2,3,4,5\right]$ is usually of the order of 10^{-3}, which at first appears to be an insignificant amount. However, it has an accumulative effect. Consider the gate in Figure 3.22. Charge is entered and transferred to well G5, which involves four clocked transfers. Thus the fraction of charge reaching G5 is:


P1
X_i
V_1
$V_1 + a_i$
ϕ_1
ϕ_2
ϕ_3
$V_1 + \frac{i-1}{k}$
ϕ_2
ϕ_3
ϕ_1
V_1
P
ϕ_s
D
E
A
B
C
D
E
A

(cont)

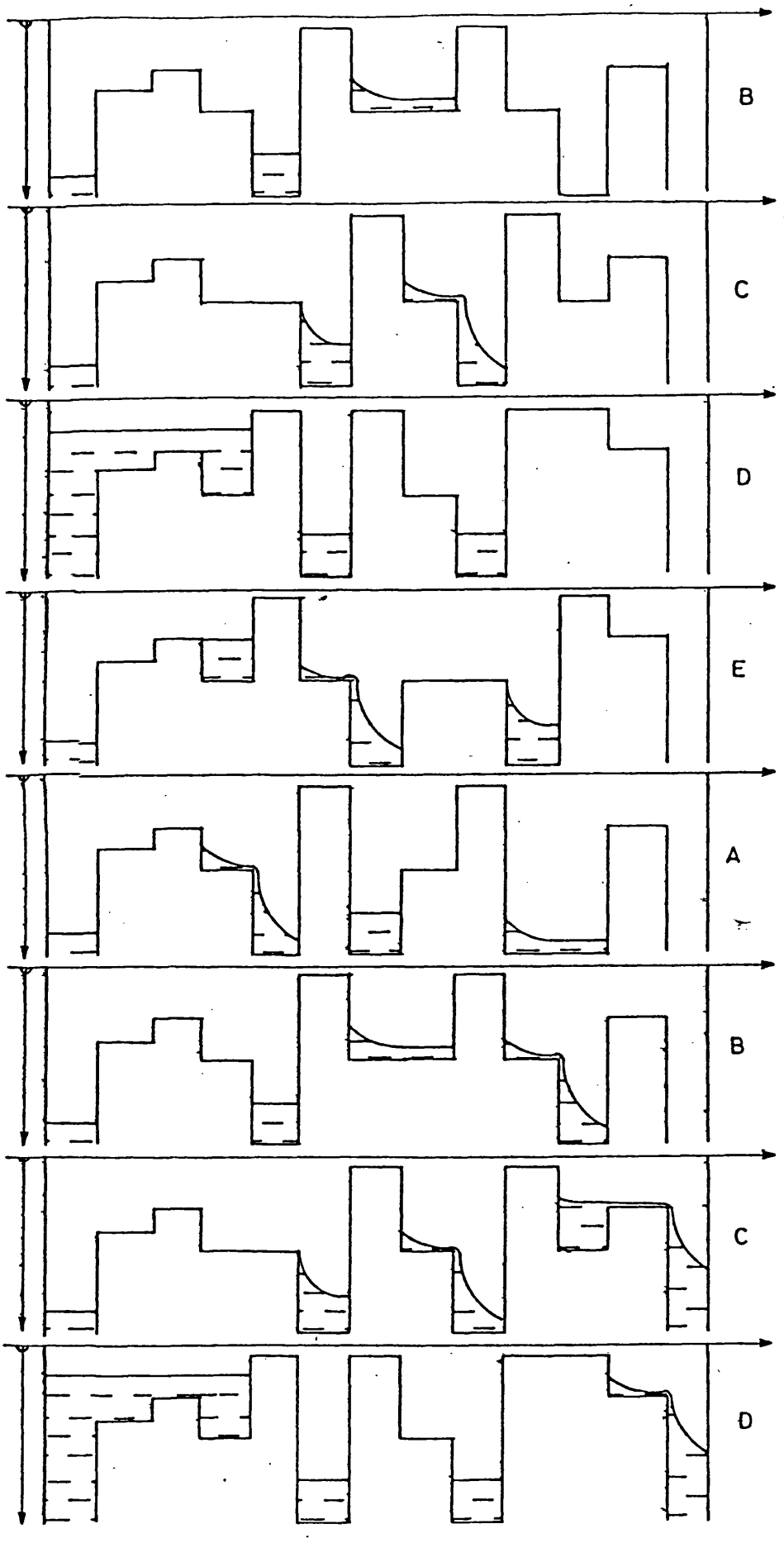


Figure 3.23 Cross section of gate showing charge transfer through the device

$$Q = (1 - \varepsilon)^4 \sum_{i=1}^{n} a_i x_i \qquad 3.8$$

Since ε is very small compared to 1, higher powers of ε can be ignored, giving an approximate value for Q as:

$$Q = (1 - 4\varepsilon) \sum_{i=1}^{n} a_i x_i \qquad 3.9$$

The worst case arises when all the input variables x_i are logic 1's, and the fifth threshold t_5 is equal to $\sum_{i=1}^{n} a_i$, so that the well G10 should contain a unit charge packet. In fact the amount will be:

$$Q = (1 - 4\varepsilon) \sum_{i=1}^{n} a_i - t_5 + 1$$

$$= 1 - 4\varepsilon t_5 \qquad 3.10$$

This charge is then transferred to the output which again involves four clocked transfers, so that the amount of charge which reaches the output instead of one charge packet is:

$$Q = 1 - 4\varepsilon - 4\varepsilon t_5 \qquad 3.11$$

Table 3.1 shows that even with $\varepsilon = 10^{-3}$, a 90% charge packet can be obtained at the output with values of t_5 and hence $\sum_{i=1}^{n} a_i$ of up to 24. This is sufficient for any four input variable function, but may be restrictive for more than four.

		Q = 0.9	Q = 0.8
ε	10^{-3}	24	49
	10^{-4}	249	499

Table 3.1 Worst case values for t_5 and hence $\sum_{i=1}^{n} a_i$

However, Table 3.1 also shows that either an improvement in the value of ε or a reduction in the amount of charge reaching the output would allow larger weights to be accommodated.

Another problem which restricts the gate is the fluctuation in the voltages applied to the wells. Consider the situation where the voltages can vary by the fractional amount $\pm \beta$. The worst case occurs when, as before, all the input variables are logic 1's, t_5 is equal to $\sum_{i=1}^{n} a_i$, and all the voltages which are proportional to the weights at the input vary by $-\beta$, and those proportional to the thresholds by $+\beta$. The charge now reaching the output instead of being one charge packet is:

$$Q = (1 - \beta).(1 - 4\varepsilon).t_5 - (1 + \beta).(t_5 - 1).(1 - 4\varepsilon)$$

$$= (1 + \beta - 4\varepsilon - 4\beta\varepsilon) + t_5(12\beta\varepsilon - 2\beta - 4\varepsilon) \qquad 3.12$$

In this instance it is found that β has to be of the order of 10^{-3} to obtain values of t_5 and hence $\sum_{i=1}^{n} a_i$ similar to the previous values as shown in Table 3.2.

		Q	
		0.9	0.8
ε	10^{-3}	16	33
	10^{-4}	42	83

Table 3.2 Worst case values for t_5 with $\beta = 10^{-3}$

Clearly this is a very difficult specification to meet. However, if the voltages which determine the weights and thresholds could be made to deviate in the same way, i.e. either all too high or all too low, then the system would be vastly improved. This could be achieved if all voltages are obtained by tapping potential dividers from the same

voltage rail. The worst case now occurs when the voltage fluctuates by the fraction $-\beta$, all other conditions being the same as before. The charge reaching the output would now be:

$$Q = (1 - \beta).(1 - 4\varepsilon).t_5 - (1 - \beta).(t_5 - 1).(1 - 4\varepsilon)$$

$$= (1 - \beta).(1 - 4\varepsilon - 4\varepsilon t_5) \qquad 3.13$$

	β	$= 10^{-2}$	$= 10^{-1}$	
	Q	= 0.9	0.882	0.8
ε	10^{-3}	24	4	27
	10^{-4}	227	49	277

Table 3.3 Worst case values of t_5

It can be seen from Table 3.3 that there is a great improvement. In fact it is possible that with $\varepsilon = 10^{-4}$ a charge packet of 88.2% can be obtained at the output with the sum of the weights having a value of up to 49, even when the voltage rail fluctuates by 10%.

3.5 Conclusions and Further Work

In comparison with other C.C.D. logic gates, it can be seen that the threshold logic gate of Figure 3.22 is similar in many ways to the binary and particularly the multi-valued full adder circuit of Figure 3.13. Thus the gate would be of a similar size and operate at similar speeds. The main differences, however, lie firstly in the fact that the threshold logic gate, although using multi-levels, does not need to detect different levels but just the presence or absence of charge; thus the output circuitry does not need to be as sophisticated as in multi-level logic. Secondly, only one threshold logic gate is required with a small number of half exclusive-or gates, whereas using binary logic on array would

be required which may need to contain a large number of gates. Therefore threshold logic provides a very compact solution which could easily be incorporated into a larger C.C.D. digital processing system.

As an individual device, however, i.e. a single chip containing a multi-threshold gate only, it would seem to be a far too complicated approach when compared to some of the other proposed threshold logic gates [18], and also much slower. In fact it is the speed aspect which severely handicaps the device as recent developments have shown [20]. The limiting factor in the speed of operation of a C.C.D. is charge overflow; the further the charge has to overflow the slower the device. Since the threshold logic gate relies heavily on this operation, particularly with multi-thresholds, this imposes a restriction on the number of thresholds allowed when a particular speed is required. However, this problem is common to all C.C.D.'s so that the threshold logic gate is still a viable alternative to the present C.C.D. logic gates.

Some alternative technological developments that have appeared of late may overcome some of the problems involved with C.C.D.'s. For example, the buried channel device, B.C.D., can improve the value of ε and the speed of operation, but this has to be paid for by a decrease in charge handling capacity which, for use in a threshold logic device, is an undesirable feature. Another idea is the use of Gallium Arsenide, GaAs, instead of Silicon, which has resulted in an increase in frequency of operation from just a few megahertz to a few hundred [21].

Finally, since the C.C.D. threshold logic gate is a clocked device, it is interesting to note that work has been done involving synchronous

threshold logic circuits [22,23]. An area of further work would therefore be to expand on this to include, for example, the role of multi-threshold logic in sequential systems. Also, there is the possibility of using feedback which may increase the power of the gate and hence open up another area of future research.

References

1. Boyle, W.S. and Smith, G.E.: "Charge coupled semiconductor devices", Bell Syst. Tech. J. Briefs, pp.588-593, Apr. 1970.

2. Sequin, C.H. and Tompsett, M.F.: "Charge Transfer Devices", (Academic Press, 1975).

3. Hobson, G.S.: "Charge-Transfer Devices", (Edward Arnold, 1978).

4. Howes, M.J. and Morgan, D.V.: "Charge-coupled Devices and Systems", (John Wiley, 1979).

5. Beynon, J.D.E. and Lamb, D.R.: "Charge-Coupled Devices and Their Application", (McGraw Hill, 1980).

6. Zimmerman, T.A. and Barbe, D.F.: "A new role for charge-coupled devices: digital signal processing", Electronics, pp.97-103, March 31, 1977.

7. Mok, T.D. and Salama, C.A.T.: "Logic array using charge-transfer devices", Electron. Lett., 8, 20, pp.495-496, Oct. 1972.

8. Handy, R.J.: "Use of C.C.D. in development of digital logic", Trans. IEEE, ED-24, 8, pp.1049-1061, Aug. 1977.

9. Zimmerman, T.A., Allen, R.A. and Jacobs, R.W.: "Digital charge-coupled logic (D.C.C.L.)", IEEE J. Solid State Circuits, SC-12, 5, pp.4731485, Oct. 1977.

10. Montgomery, J.H. and Gamble, H.S.: "Basic C.C.D. logic gates", The Radio and Electronic Eng., 50, 5, pp.258-268, May 1980.

11. Yamada, M., Fiyishima, K., Nagasawa, K. and Gamou, Y.: " A new multilevel storage structure for high density C.C.D. memory", IEEE J. Solid State Circuits, SC-13, 5, pp.688-692, Oct. 1978.

12.. Kerkhoff, H.G. and Kijkstra, H.: "The application of C.C.D.'s in multiple-valued logic", Proc. 5th International Conference on Charge Coupled Devices, (Edinburgh), pp.304-309, 1979.

13. Kerkhoff, H.G. and Tervoert, M.L.: "The implementation of multiple-valued functions using charge-coupled devices", Proc. 10th International Symposium on Multi-Valued Logic, pp.6-15, 1980.

14. Kerkhoff, H.G. and Tervoert, M.L.: "Multiple-valued logic charge-coupled devices", Trans. IEEE, C-30, 9, pp.644-653, Sept. 1981.

15. Van Roermund, A. and Coppelmans, P.: "A circulating multilevel charge transfer device memory with adaptive refresher", Proc. 5th International Conference on Charge Coupled Devices, (Edinburgh), pp.310-316, 1979.

16. Mavor, J.: "M.O.S.T. Integrated-Circuit Engineering", (Peter Peregrinus, 1973).

17. Edwards, C.R.: "Some novel exclusive-OR/NOR circuits", Electron. Lett., 11, 1, pp.3-4, Jan. 1975.

18. Hurst, S.L.: "Logical Processing of Digital Signals", (Crane-Russak, N.Y., and Edward Arnold, London, 1978).

19. Picton, P.D.: "The realisation of multi-threshold threshold logic gates using charge-coupled devices", Proc. IEE, Pt.E, 3, pp.107-110, May 1982.

20. Kerkhoff, H.G., Tervoert, M.L. and Tilmans, H.A.C.: "Design considerations and measurement results of multiple-valued logic C.C.D.'s", Proc. 11th International Symposium on Multiple-Valued Logic, pp.205-211, May 1981.

21. Deyhimy, I., Anderson, R.J., Eden, R.C. and Harris Jr., J.S.: "Charge-coupled devices in gallium arsenide", Proc.IEE, Pt.1, 127, 5, pp.278-287, Oct. 1980.

22. Masters, G.M. and Mattson, R.L.: "The application of threshold logic to the design of sequential machines", Proc. 1966 Annual Symposium on Switching and Automation Theory, pp.184-194, Oct. 1966.

23. Hadlock, F.O. and Coates, C.L.: "Realisation of sequential machines with threshold elements", Trans. IEEE, C-18, 5, pp.428-439, May 1969.

24. Grove, A.S.: "Physics and Technology of Semiconductor Devices", (John Wiley, 1967).

CHAPTER 4

SERIAL INPUT LOGIC

4. Serial Input Logic

4.1 General and Desirable Features

Recent work has indicated a need for the efficient high-speed processing of logic data where the input and output are in a word-formatted serial stream [1,2]. Figure 4.1 shows a schematic diagram of such a system.

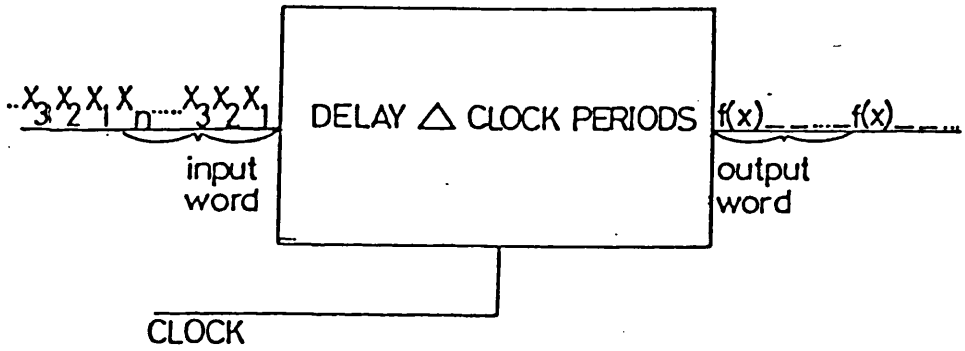


Figure 4.1 Schematic representation of a serial input logic system

The input to the system receives an n bit word, each bit corresponding to the logical state of one of the input variables x_i, and after a delay of Δ clock periods since the last bit x_n enters the system, yields an output function f(x) of the input word. The remainder of the bits in the output word are don't care terms since they are unwanted signals.

The most obvious method of realising this type of system is to initially perform a serial to parallel conversion using a shift register and then processing the data using conventional combinational logic circuitry as in Figure 4.2.

Since the data arrives at the system input at a constant rate, with a clock period of T, say, in order that the output is not corrupted by the end of the previous input word or the beginning of the next

input word, T must be greater than or equal to the propagation delay time of the combinational circuit. Thus although the delay Δ of the system is only one clock period in this instance, that clock period must be greater than the propagation delay of the combinational logic circuitry, which firstly varies according to whatever function is being realised, and secondly, for large values of n, this delay could be large, thus slowing down the system.

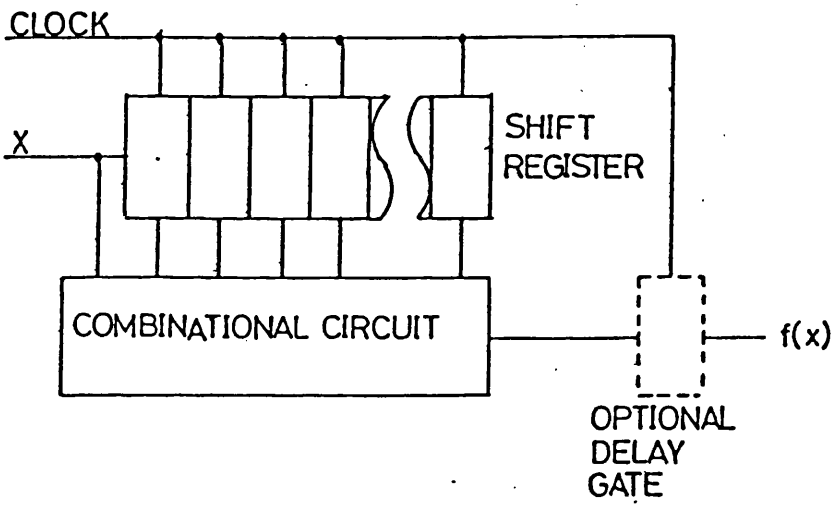


Figure 4.2 Serial to parallel conversion

It is desirable, therefore, for a serial input system to have the following features:

a) no serial to parallel conversion; the data is processed as it arrives at the input,

b) no resetting between input words thus saving one clock period delay,

c) the delay Δ should be kept to a minimum, i.e. one,

d) the logical hardware should be uniform, e.g. universal logic modules, so that the delay at each stage is well defined and if possible minimal.

To date the only system that has been developed which approaches these four ideals is mode-controlled logic [1,2] which will be discussed in detail later in this chapter. A more general approach to obtaining a system which achieves the above features will now be considered.

4.2 Conventional Sequential Logic Realisation

A general sequential logic system is shown in Figure 4.3. The next state N is a function of the present state S and the input x, whereas the output Z can be either a function of the present state only, in which case the system is known as the Moore model, or the present state and the input, in which case it is known as the Mealy model [7]. In the following discussion and throughout the chapter the Moore model only will be used.

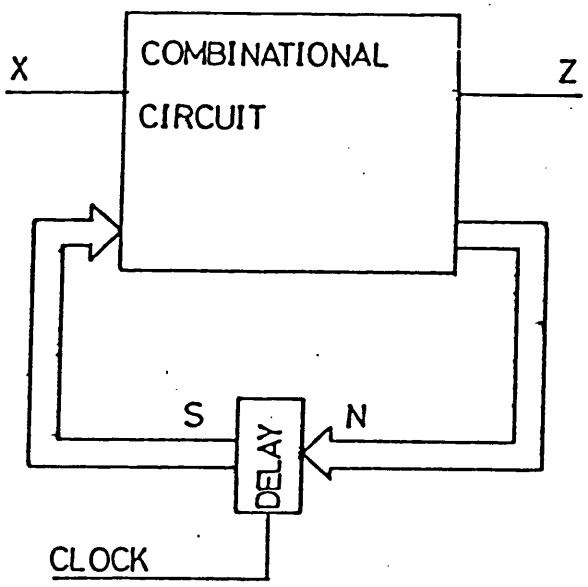


Figure 4.3 General sequential logic system

Typically a sequential machine would be realised by the following design processes:

a) draw a flow diagram of the required system,

b) draw a state table,

c) reduce the number of states to give a minimal solution,

d) assign logical values to the states,

e) determine the hardware realisation.

It is the third step, state minimisation, that is of particular interest and will be discussed in greater depth.

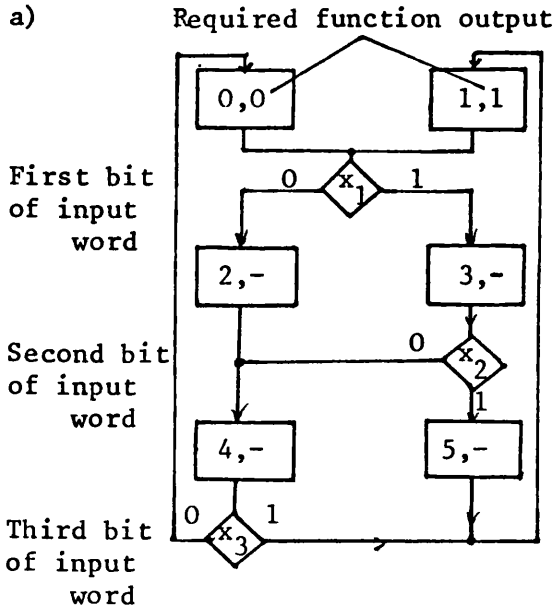
4.2.1 State Reduction in the Realisation of a Specific Function

Consider the logical function given in equation 4.1.

$$f(x) = x_1 \cdot x_2 + x_3 \qquad 4.1$$

Figure 4.4(a) and (b) show the flow diagram and state table respectively of a system which realises this function with serial inputs.

a)



b)

x = 0 , 1			
S	N	N	Z
0	2	3	0
1	2	3	1
2	4	4	-
3	4	5	-
4	0	1	-
5	1	1	-

Figure 4.4 Serial input sequential system
(a) Flow diagram for $f(x) = x_1 \cdot x_2 + x_3$
(b) State table for the same function

It can be seen from the flow diagram that if the machine starts in either state 0 or state 1 then the function f(x) is produced at the output immediately after the arrival of the input variable x_3. Therefore, the first three ideal characteristics of a serial input processor have been achieved if this flow diagram can be realised. Also in the state table it can be seen that there are a large number of don't care terms in the output column which is useful in the reduction of the number of states.

The usual method used for reducing the number of states of a machine [5,6,7,8,9] involves the use of compatible pairs of states which are defined as follows:

Definition 4.1

A pair of states S_i and S_j are said to be output compatible, shown as $S_i \sim S_j$, if the outputs of these two states, namely Z_i and Z_j, are identical in a completely specified machine, or if one or both outputs are don't care terms in an incompletely specified machine. If they are compatible then the compatible pair $C = S_iS_j$ can be formed.

Whether or not pairs of states are compatible usually depends on whether or not the pairs of next states for all input values are also compatible. These pairs of next states are said to be implied by the original compatibles and are therefore known as the implied compatibles.

The set of primary implied compatibles, PC's, can be defined as follows:

Definition 4.2

A compatible C_i has a set of primary implied compatibles PC_i which consists of all compatibles C_{ij} implied by C_i for an input x_j such that:

i) C_{ij} has more than one element,

ii) $C_{ij} \not\subset C_i$

iii) $C_{ij} \not\subset C_{ik}$ if $C_{ik} \in PC_i$

Incompatibility can now be defined as follows:

Definition 4.3

A pair of states S_i and S_j are said to be incompatible, shown as $S_i \not\sim S_j$, if their outputs, namely Z_i and Z_j, are not equal, i.e. one is a logic 0 and the other is a logic 1, or if at least one of their primary implied compatible pairs of states is incompatible.

Sets of compatible states can be formed by combining compatible pairs.

Definition 4.4

A set of states $C = S_1S_2...S_k$ from a state table of a machine M is a compatible set if and only if for every pair of states $S_i, S_j \in C$, $S_i \sim S_j$..

In the case of completely specified machines the compatibility function is transitive i.e. if $S_i \sim S_j$ and $S_j \sim S_k$, then it follows that $S_i \sim S_k$ and the compatible $C = S_iS_jS_k$ can be formed. For incompletely specified machines this is not the case and so in order to form the compatible $C = S_iS_jS_k$, all three pairs of states have to be shown to be compatible.

Sets of maximal compatibles, MC's, can be formed where they are defined as follows:

Definition 4.5

A maximal compatible is a compatible which is not a subset of any other compatible.

A method for determining all the compatible pairs is the "pair chart" as shown in Figure 4.5.

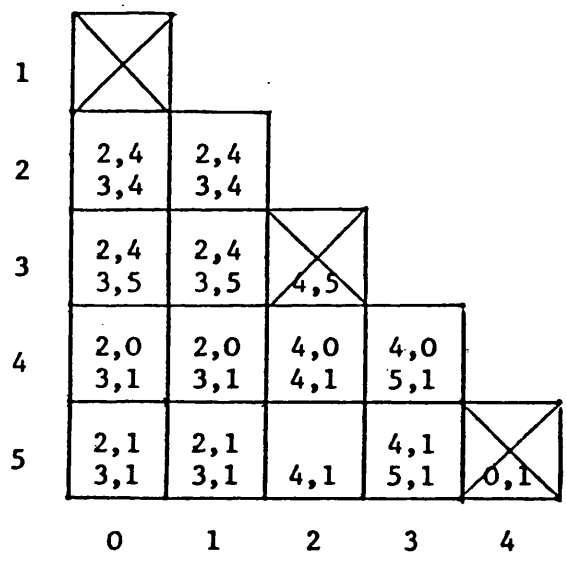


Figure 4.5 Pair chart

The pair chart is drawn such that every pair of states are examined, and written in their corresponding boxes are the full sets of primary implied compatibles for each pair. If a pair of states are incompatible then a cross is placed over the corresponding box, e.g. the pair of states 0 and 1 are incompatible as can be seen from the state table of Figure 4.4(b) since they have opposite outputs. Using definition 4.3, it can be seen that 4 $\not\sim$ 5 because 0 $\not\sim$ 1 and hence 2 $\not\sim$ 3. The remaining compatibles are:

02,03,04, 12,13,14,24,25,34,35

Combining compatible pairs, the maximal compatibles can be formed, e.g. the three pairs 02, 04, and 24 can be combined to give 024. The maximal compatibles are therefore:

024,025,034,035,124,125,134,135

Note that a compatible pair can be used to form more than one of the maximal compatibles since the machine is incompletely specified. Had it been completely specified then this would not be the case and therefore the maximal compatibles would have been disjoint.

In order to obtain the final solution, a minimal closed cover is looked for where its states can be selected from the original states or from the sets of compatible states including the maximal compatibles.

Definition 4.6

A machine M having states S is said to be covered by a machine M' with states S' if every state in S is a subset of at least one of the states in S'.

Definition 4.7

A set of compatibles is said to be closed if every compatible in the set has its primary implied compatibles also contained in at least one of the compatibles in the set.

Definition 4.8

If a machine M can be replaced by a machine M', then M' is said to be minimal if every other machine M'' that can replace M and M itself has more states than M'.

For a completely specified machine the set of maximal compatibles form the minimal closed cover and therefore the original machine can be replaced by a new machine whose states are the maximal compatibles. For an incompletely specified machine the situation is far more complex; not only have the maximal compatibles to be considered but all subsets including the single states.

Definition 4.9

The set of prime compatibles consists of all maximal compatibles and all subsets of maximal compatibles which are not dominated by any other compatibles.

Definition 4.10

A compatible C_i is said to dominate another compatible C_j if $C_i \supset C_j$ and $PC_i \subseteq PC_j$ where PC_i and PC_j are the primary implied compatibles as defined in definition 4.2.

For the example under consideration the full set of prime compatibles is shown in Table 4.1.

Included in the table are the further implied compatibles which are the compatibles implied by the primary implied compatibles and then those implied by the first set of further implied and the process repeated to give the full closure class set.

Definition 4.11

The closure class set E_i of a compatible C_i is a set of all compatibles implied by C_i obtained by repeated use of the transitivity of implication such that the compatibles which are subsets of either C_i or any other member of E_i are removed from the set.

Prime compatibles	Closure class set: Primary implied compatibles	Closure class set: Further implied compatibles	Implication condition
024	134	135,124	I
025	124,134	024,135	U
034	024,135	134,124	U
035	124,135	024,134	U
124	024,134	135	I
125	124,134	024,135	U
134	024,135	124	I
135	124	024,134	I
02	24,34	04,14,15,13,12,35	I
03	24,35	04,14,15,02,12,13,34	U
04	02,13	24,34,35,14,15,12	I
05	12,13	24,34,35,04,14,15,02	U
12	24,34	04,14,15,13,12,35	I
13	24,35	04,14,15,02,12,34	I
14	02,13	24,34,35,14,15,12	I
15	12,13	24,35,34,04,14,02	I
24	04,14	02,13,34,35,15,12	I
25	14	02,13,24,34,35,04,15,12	U
34	04,15	02,13,12,24,35,14	I
35	14,15	02,13,12,24,04,34	I
0	-	-	-
1	-	-	-
2	-	-	-
3	-	-	-
4	-	-	-
5	-	-	-

Table 4.1 Full set of prime compatibles and their implied closure set

As an example consider the compatible C_1 = 024. From the state table of Figure 4.4(b) it can be found that the implied compatibles are 024 and 134, but by definition 4.11 the former is removed because it is a subset of C_1. The compatible C_2 = 134 further implies 135 and 024, the latter again being removed, which in turn implies 124 and 135, 135 being removed because it has already appeared. The compatible 124 implies 024 and 134, both of which can be removed and so the implication process is terminated, and the full closure class set has been found.

Also included in the table is the information about whether a compatible is implied or not, marked I or U respectively, i.e. whether or not the compatible is a subset of at least one of the compatibles contained in any other compatible's closure class set. This can then be used to remove some more prime compatibles from the table by the use of the following definition:

Definition 4.12

If a compatible is unimplied and at least one of its states is included in at least one of the compatibles of its closure class set then that compatible can be eliminated from the set of prime compatibles [6].

In Table 4.1 all the unimplied compatibles can be eliminated by this definition, and a minimal closed cover can now be chosen from the remaining implied compatibles. It is interesting to note that all r^{th} ordered compatibles, i.e. those that have r states, imply closure class sets of only r^{th} ordered compatibles. It would be futile therefore to form a cover with mixed order compatibles because the highest ordered ones in that cover would form a closed cover by themselves. Thus all closed covers have to contain only compatibles of

the same order, which in this instance means that there are three possible covers:

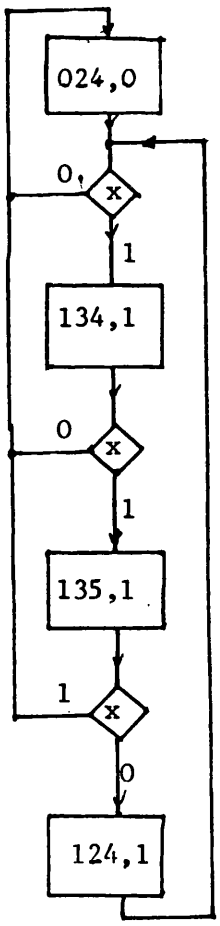
a) 024,124,134,135

b) 02,14,12,13,14,15,24,34,35

c) 0,1,2,3,4,5

Clearly the set of maximal compatibles is the minimal closed cover since it has only four elements. The new flow diagram and state table for this machine is shown in Figure 4.6(a) and (b), where it can be seen that the input x is now not labelled with x_1, x_2 and x_3 since it no longer matters what state the machine starts in.

a)



b)

x = 0 , 1

S	N	N	Z
024	024	134	0
124	024	134	1
134	024	135	1
135	124	135	1

Figure 4.6(a) Reduced flow diagram for function $f(x) = x_1.x_2 + x_3$
(b) State table for the same function

It has therefore been shown that the minimal closed cover for this particular function is composed of the implied maximal compatibles. This is due to three reasons:

i) every r^{th} order compatible implies r_{th} order compatibles so that the closure class sets contain compatibles of the same order only,

ii) every unimplied compatible could be eliminated from the set of prime compatibles using definition 4.12,

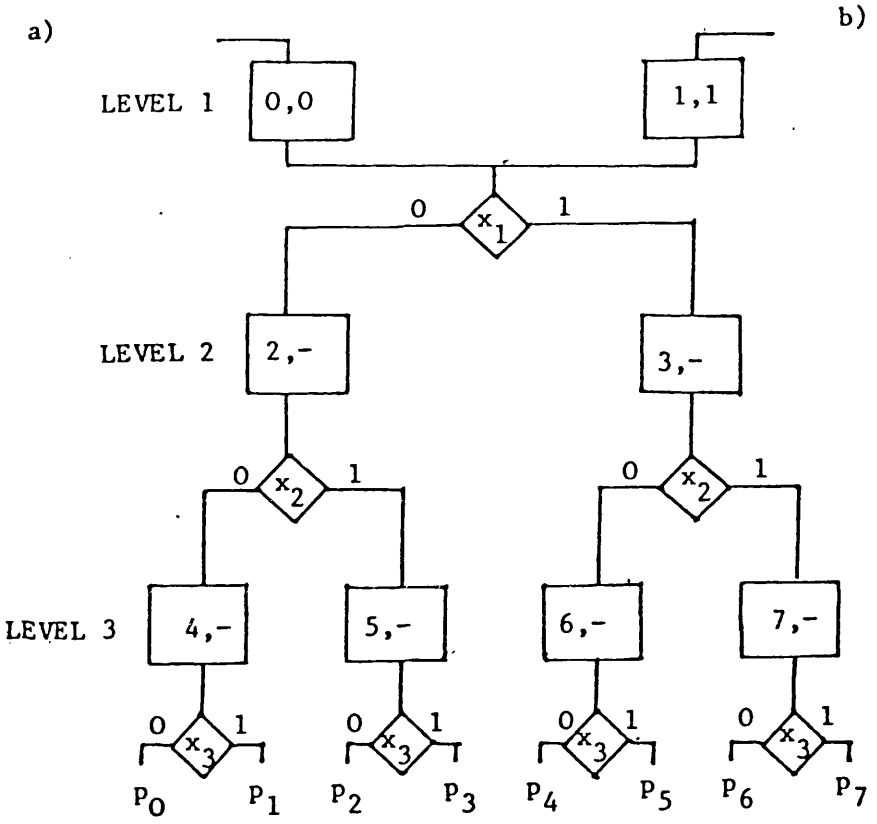
iii) although as yet unstated it can be seen that every r^{th} order closed cover is the complete set of r^{th} order subsets of the maximal compatibles cover and hence contains more elements.

If these three conditions can be shown to occur for any function then the problem of state minimisation is greatly reduced and would consist of merely finding the set of implied maximal compatibles, and comparing their number with that of the single states of the original machine. If this number is less, then the maximal compatibles become the states of the new reduced machine, and if more then the machine cannot be reduced and therefore stays as it is. However, there may be some advantage in using the maximal compatible machine even if there are more states, because the restriction of the machine starting in particular states would be eliminated.

4.2.2. State Reduction for a General Function

Figure 4.7(a) shows the flow diagram for any three input variable function. States 4, 5, 6 and 7 all branch to either state 0 or state 1 dependent on the function being processed and so the labels p_i are

a)



b)

$x_1 x_2 x_3$	f(x)
0 0 0	p_0
0 0 1	p_1
0 1 0	p_2
0 1 1	p_3
1 0 0	p_4
1 0 1	p_5
1 1 0	p_6
1 1 1	p_7

c) x = 0 , 1

S	N	N	Z
0	2	3	0
1	2	3	1
2	4	5	-
3	6	7	-
4	p_0	p_1	-
5	p_2	p_3	-
6	p_4	p_5	-
7	p_6	p_7	-

d)

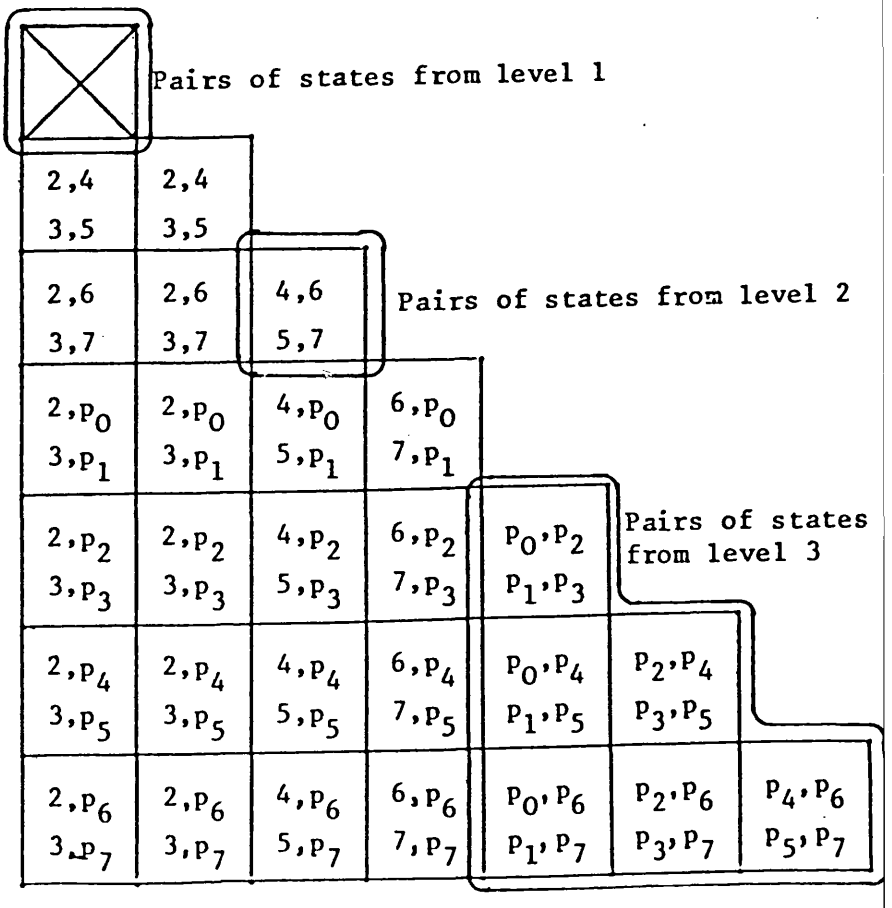


Figure 4.7 (a) General flow diagram for any function of n = 3
(b) Truth table
(c) State table
(d) Pair chart

included which can equal 0 or 1 and correspond to the function at all the minterm positions as shown in the truth table of Figure 4.7(b). The state table of Figure 4.7(c) also illustrates this point and can be used to form the pair chart shown in Figure 4.7(d).

If this pair chart is examined in detail it is noticed that the only possible incompatible pairs of states are those on the same level, states from different levels being always compatible. In particular whether states 4, 5, 6 and 7 are compatible or not is dependent on the p_i values, e.g. states 6 and 7 are compatible if $p_4 = p_6$ and $p_5 = p_7$, in which case their set of primary implied compatibles is empty, or they are incompatible if $p_4 \neq p_6$ or $p_5 \neq p_7$ since then they would imply the compatible C = 01 which is already shown to be incompatible.

Furthermore, among these four states compatibility is transitive, e.g. if $4 \sim 5$ and $4 \sim 6$ then for this to be so it must be because $p_0 = p_2$, $p_1 = p_3$, $p_0 = p_4$ and $p_1 = p_5$. It therefore follows that $p_2 = p_4$ and $p_3 = p_5$ which satisfies the conditions for $5 \sim 6$. Finally, it can be shown that if two of these states, S_i and S_j say are compatible then no prime compatible will exist which has only S_i or only S_j in it. This is because from definition 4.10, any prime compatible which contains only S_i or only S_j is a subset of the prime compatible which contains the same states but with both S_i and S_j, and since the compatible pair S_iS_j implies an empty set, both these prime compatibles would imply the same compatibles so that the latter would dominate the former which could therefore be removed.

These features are similar to those that can be found in completely specified machines which enables the equivalence relationship [8] to be used. Thus, of the states 4, 5, 6 and 7, if any two, S_i and S_j, say,

are compatible then they can also be said to be equivalent, i.e. $S_i = S_j$.

States other than 4, 5, 6 and 7 but which are on the same level as each other imply states which are also on the same level as each other, which ultimately imply some of the states 4, 5, 6 and 7. As the equivalence operation is transitive it follows that any states which are on the same level and which are compatible are also equivalent.

Essentially, what this equivalence operation does to the flow diagram of Figure 4.7(a) is to merge together into one state any states on the same level which are found to be compatible. If the flow diagram of Figure 4.4(a) is examined and compared to the general flow diagram, then it can be seen that the three states 4, 5 and 6 have been merged into one state 4, because each of these three states, upon the arrival of the variable x_3, branch in the same manner to states 0 and 1 and are therefore equivalent.

This equivalence operation can be put to good use in the derivation of the maximal compatibles which in this instance uses the product-of-sums method [10]. Essentially this is done by taking every incompatible pair of states, S_i and S_j say, forming the logic sum $(S_i + S_j)$, and then taking the logical product of all these sums. Expanding this expression yields a sum-of-products, each product term being used to derive a maximal compatible by removing the states in the product term from the full set of states. As an example, the pair chart of Figure 4.5 shows that there are three incompatible pairs, namely 01, 23 and 45. These can be used to form the product-of-sums as in equation 4.2.

$$(0 + 1).(2 + 3).(4 + 5) \qquad 4.2$$

Expanding gives:

$$024 + 025 + 034 + 035 + 124 + 125 + 134 + 135 \qquad 4.3$$

Each product term is then used to obtain the maximal compatibles by subtracting it from the full set 012345. Thus:

Product terms		Maximal compatibles
024	gives	135
025	gives	134
034	gives	125
035	gives	124
124	gives	035
125	gives	034
134	gives	025
135	gives	024

Table 4.2

Comparing with Table 4.1 it is found that the maximal compatibles are identical.

A simpler method of presenting the maximal compatibles, rather than to list them, is to express them also in a product-of-sums. Thus the above set of maximal compatibles can be summarised as the expression given in equation 4.4.

$$(0 + 1).(2 + 3).(4 + 5) \qquad 4.4$$

It is a coincidence that for this example equations 4.2 and 4.4 are identical, this not being the usual case. Returning to the general case, it has been said that the only possible pairs of states that can

be incompatible are those on the same level. Thus consider the situation where all of these are incompatible, giving the product-of-sums as:

$$(0 + 1).(2 + 3).(4 + 5).(4 + 6).(4 + 7).(5 + 6).(5 + 7).(6 + 7) \qquad 4.5$$

Partially expanding gives:

$$(0 + 1).(2 + 3).(456 + 457 + 467 + 567) \qquad 4.6$$

If this expression is fully expanded, the maximal compatibles found from the product terms and then reorganised into a product-of-sums expression as done previously, then equation 4.7 is obtained.

$$(0 + 1).(2 + 3).(4 + 5 + 6 + 7) \qquad 4.7$$

Note that each bracket contains the sum of the states on any particular level. If, instead of all pairs of states on the same level being incompatible, some are compatible, e.g. $6 \sim 7$, then it has been shown that $6 = 7$, and thus the expression for the maximal compatibles becomes:

$$(0 + 1).(2 + 3).(4 + 5 + 6 + 6) = (0 + 1).(2 + 3).(4 + 5 + 6) \qquad 4.8$$

since $6 + 6 = 6$.

This expression is the same as would have been obtained if the states 6 and 7 were first merged into one state called state 6 and then the maximal compatibles obtained. Thus in the example in the previous section, it has been said that it is equivalent to the general solution but with the states $4 = 5 = 6$. Substituting this into equation 4.7 gives:

$$(0 + 1).(2 + 3).(4 + 4 + 4 + 7) = (0 + 1).(2 + 3).(4 + 7) \qquad 4.9$$

This expression agrees with that originally found in equation 4.4 except that state 5 has been called state 7 which is a trivial difference that arises simply because of the initial state labelling, and does not affect the solution.

Therefore the general expression for the maximal compatibles comprises of the logical product-of-sums, where each sum consists of all the states on any particular level, which may then be found to be equivalent and hence removed from the term. Since each maximal compatible is found by the expansion of the product-of-sums expression their order must be equal to the number of bracketed terms which, since each represents the states on a particular level, must equal the number of levels. However, the number of levels arises because of the number of input variables that are being processed, n, and hence the order of the maximal compatibles is n. Furthermore, it can be seen from the flow diagram of Figure 4.7(a) that states on level i, say, imply or have next states on level i + 1 except for the very last level which returns to the first level. Thus each bracketed term in the product-of-sums expression for the maximal compatibles implies states from a different bracketed term.

The term (0 + 1) implies states contained in the term (2 + 3)

" (2 + 3) " " (4 + 5 + 6 + 7)

" (4 + 5 + 6 + 7) " " (0 + 1)

Hence no states on different levels can imply states on the same level and it follows that if the maximal compatibles are of the order n, then they imply compatibles of the order n. In fact this applies to all compatibles of any order r, i.e. all r^{th} order compatibles imply only r^{th} order compatibles, so that the first condition for a general

solution consisting of maximal compatibles, as stated at the end of section 4.2.1, has been met.

The next condition to be determined is whether or not all the unimplied compatibles can be removed from the set of prime compatibles using definition 4.12.

It is possible to derive the implied maximal compatibles directly from the general expression of equation 4.7 using the state table of Figure 4.7(c) as follows:

States in (0 + 1) have next states in (2) when x is a 0 and (3) when x is a 1

" (2 + 3) " (4 + 6) " (5 + 7) "

" (4 + 5 + 6 + 7) " $(p_0 + p_2 + p_4 + p_6)$ " $(p_1 + p_3 + p_5 + p_7)$ "

Thus the expression for the primary implied maximal compatibles is:

$$(2).(4 + 6).(p_0 + p_2 + p_4 + p_6) + (3).(5 + 7).(p_1 + p_3 + p_5 + p_7) \qquad 4.10$$

The terms p_i are either 1's or 0's and so it can be seen that the set of maximal compatibles contained in equation 4.10 are a subset of those contained in equation 4.7. The maximal compatibles not contained in equation 4.10 are therefore some of the unimplied compatibles and since all the states are contained in equation 4.10, by definition 4.12 the unimplied compatibles can be deleted from the set of prime compatibles.

The maximal compatibles in equation 4.10 further imply compatibles as follows:

States in (2) have next states in (4) when x is a 0 and (5) when x is a 1

" (4 + 6) " $(p_0 + p_4)$ " $(p_1 + p_5)$ "

" $(p_0 + p_2 + p_4 + p_6)$ " (2) " (3) "

" (3) " (6) " (7)

" (5 + 7) " $(p_2 + p_6)$ " $(p_3 + p_7)$ "

" $(p_1 + p_3 + p_5 + p_7)$ " (2) " (3) "

Thus the expression for the first set of further implied compatibles is:

$$(p_0 + p_4).(2).(4) + (p_1 + p_5).(3).(5) + (p_2 + p_6).(2).(6) + (p_3 + p_7).(3).(7) \qquad 4.11$$

The set of compatibles implied by this set turns out to be the same, so that no further implication is needed, and therefore the maximal compatibles contained in this expression form a closed cover. The compatibles not contained in this set are the unimplied maximal compatibles, and for the same reason as before can be eliminated from the set of prime compatibles.

Lower ordered compatibles are also found to be the same in that all unimplied compatibles can be eliminated using the same process as above. As a result, Table 4.3 can be compiled which contains all the implied prime compatibles for a general function of n = 3.

It can be seen from Table 4.3 that the second order compatibles are the full second order subset of the maximal compatibles. In fact it can be shown that all implied compatibles are subsets of the implied maximal compatibles as follows:

Let a maximal compatible $MC_1 = S_1S_2 \ldots S_n$ be implied by a maximal compatible $MC_2 = S_1'S_2' \ldots S_n'$, where the subscripts on the states refer to the level that those states are on. Then:

$$S_1'S_2' \ldots S_n' \sim S_1S_2 \ldots S_n \qquad 4.12$$

Since every state implies a state on a different level it can be said that:

$$S_i' \sim S_{i+1} \quad \text{for } i = 1 \text{ to } n-1 \qquad 4.13$$

and $S_n' \sim S_1$

3^{rd} order maximal compatibles	2^{nd} order compatibles	1^{st} order compatibles or single states
p_0^{24}	p_0^{2}	0
p_1^{35}	p_0^{4}	1
p_2^{26}	p_1^{3}	2
p_3^{37}	p_1^{5}	3
p_4^{24}	p_2^{2}	4
p_5^{35}	p_2^{6}	5
p_6^{26}	p_3^{3}	6
p_7^{37}	p_3^{7}	7
	p_4^{2}	
	p_4^{4}	
	p_5^{3}	
	p_5^{5}	
	p_6^{2}	
	p_6^{6}	
	p_7^{3}	
	p_7^{7}	
	24	
	26	
	35	
	37	

Table 4.3 Implied prime compatibles for general function of n = 3

Every subset of MC_1 must therefore also be implied by the same ordered subset of MC_2, e.g.:

$$S_1'S_2' \sim S_2S_3 \qquad 4.14$$

Thus every subset of every implied maximal compatible must also be an implied compatible. This means that all three conditions required for the minimal closed cover consisting of the implied maximal compatibles have been achieved which greatly reduce the amount of work required to find a reduced system.

Returning to the example originally used, i.e. $f(x) = x_1.x_2 + x_3$, if the p_i values in the maximal compatibles in Table 4.3 are assigned according to this function, and bearing in mind that states 4, 5 and 6 are equivalent, then Table 4.4 can be drawn.

Comparing with the original solution it is found that the maximal compatibles are the same.

General implied maximal compatibles	Specific implied maximal compatibles for $f(x) = x_1.x_2 + x_3$
p_0 24	024
p_1 35	134
p_2 26	~~024~~
p_3 37	137
p_4 24	~~024~~
p_5 35	~~134~~
p_6 26	124
p_7 37	~~137~~

Table 4.4 Implied maximal compatibles, where terms are crossed out if they appear more than once, thus leaving single representatives of each implied maximal compatible

All arguments that have been presented apply equally well to functions of any value of n. Thus a new class of machines has been obtained whereby, if the number of states can be reduced, the new machine would contain states which are the implied maximal compatibles obtained from the original machine. If the number of states cannot be reduced, i.e. the number of states in the original machine is less than the number of implied maximal compatibles, then it still may be advantageous to chose the latter since it dispenses with the condition that the machine has to start in either of the states 0 or 1. Previously there were only two classes of machine which had minimal closed covers consisting of maximal compatibles, namely:

i) fully specified machines in which it consists of all the maximal compatibles,

ii) another class defined by McCluskey [11,12] in which every cover consisting of maximal compatibles is closed.

The serial input machines, therefore, form an interesting third class.

4.2.3 State Assignment and Hardware Realisation

Having reduced the number of states of the machine, they now have to be assigned binary values in order to obtain a hardware realisation. In this respect a serial input logic system is no different from any other sequential system and therefore state assignment will not be discussed but can be found in a number of texts [8,9,13]. However, it has been said earlier that one of the ideal characteristics of a serial input system is that it consists of uniform logic hardware. Thus in the diagram of Figure 4.3 it is desirable that the combinational logic consists of uniform logic types, e.g. threshold logic gates [14],

multiplexers [15], universal logic modules etc. Techniques have been developed which enable such realisations to be obtained using the Rademacher-Walsh spectrum discussed in Chapter 2. However, a feature which is overlooked in these methods is that there is a choice of delay gate or bistable used in the system. The four possible types of bistable, their characteristic equations and their spectral equivalent will therefore be briefly discussed.

4.2.3.1 Characteristic Equations of Bistables [16]

The general schematic diagram of a bistable is shown in Figure 4.8, where the next state q_i is a function of the present state y_i and the inputs q_i' and q_j', and appears upon the arrival of the clock pulse. This function is known as the characteristic function of the bistable and is listed for each type of bistable in Table 4.5.

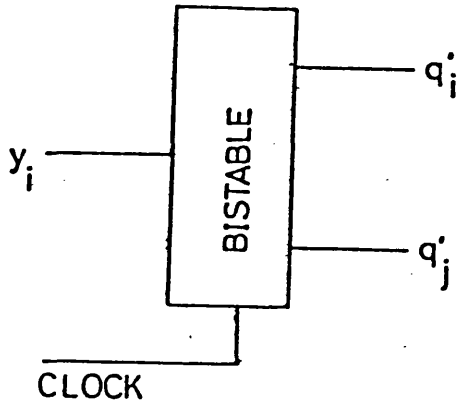


Figure 4.8 General bistable

Type of bistable	Input connections	Characteristic equation
D	$q_i' = q_j' = D$	$q_i = D$
J-K	$q_i' = J \quad q_j' = K$	$q_i = \overline{y}_i.J + y_i.\overline{K}$
R-S	$q_i' = S \quad q_j' = R$	$q_i = \overline{R}.(S + y_i)$ where R and S are disjoint
T	$q_i' = q_j' = T$	$q_i = y_i \oplus T$

Table 4.5 Characteristic equation of bistables

D bistable

Usually it is assumed that, due to its simplicity, the D-type bistable is used in which case the next state q_i is obtained from the next state function as in equation 4.15.

$$q_i = D = f_i\ (y_1, y_2, \ldots, y_m, x) \qquad 4.15$$

where m is the number of bistables in the system.

The hardware realisation could then be obtained by finding the combinational logic solution for each of the next states q_i. However, it can be seen from Table 4.5 that if any other type of bistable is used the hardware to be realised would be quite different. In fact if the spectrum of the function in equation 4.15 is as in expression 4.16, then it is interesting to determine the spectra of the other functions J,K,R,S and T.

$$r_0\ r_1\ r_2\ r_3 \cdots r_{m+1}\ r_{12}\ r_{13} \cdots r_{123..mm+1} \qquad 4.16$$

Each of the 2^{m+1} coefficients in equation 4.16 are obtained using the Rademacher-Walsh transform as described in Chapter 2. There are m+1 variables because from equation 4.15 it can be seen that there are m present states plus one input variable x which can be regarded as the variable y_{m+1}.

J-K bistable

It has been shown [17] that if a function f(x) is decomposed about one of its input variables x_i, as in equation 4.17, then the spectral coefficients of the two sub-functions are related to the spectral coefficients of the original function, r_j, as in equation 4.18.

$$f(x) = \bar{x}_i . f_1(x) + x_i . f_2(x) \qquad 4.17$$

Coefficients of $f_1(x)$ are: $\frac{1}{2}(r_j + r_{ij})$ for all subscripte j that do not contain i

" $f_2(x)$ " : $\frac{1}{2}(r_j - r_{ij})$ " " "

4.18

Comparing with the characteristic equation for a J-K bistable it can be seen that $J = f_1(x)$ and $K = \overline{f_2(x)}$. Thus the spectral coefficients for J and K are:

$$J_j = \tfrac{1}{2}(r_{ij} + r_j)$$
$$K_j = -\tfrac{1}{2}(r_j - r_{ij}) = \tfrac{1}{2}(r_{ij} - r_j) \qquad 4.19$$

for all subscripts j that do not contain i.

R-S bistable

Since, in the characteristic equation of the R-S bistable it is stated that R and S are disjoint, the equation can be rearranged to give:

$$q_i = \bar{y}_i . S + y_i . \bar{R} \qquad 4.20$$

where R and S are disjoint.

Thus, comparing with the characteristic equation of the J-K bistable it can be seen that if J and K are disjoint the equations are identical and the spectral coefficient values stated in equation 4.19 can be used for S and R. One way of testing whether J and K are disjoint or not is as follows:

If J and K are disjoint their spectral coefficients obey the following equation [16]:

$$\frac{1}{2^m}\left[J_0\ J_1\ J_2\ \ldots\ J_{12\ldots m+1}\right]\cdot\begin{bmatrix}K_0\\K_1\\K_2\\\vdots\\K_{12\ldots m+1}\end{bmatrix} = -2^m + J_0 + K_0 \qquad 4.21$$

Substituting from equation 4.19 gives:

$$\sum_{\substack{\text{for all } j\\ \text{not}\\ \text{containing } i}}\left[r_{ij}^2 - r_j^2\right] = -2^{2(m+1)} + 2^{m+2}.r_i \qquad 4.22$$

But it has been shown that [18] the sum of the squares of the spectral coefficients is $2^{2(m+1)}$, thus:

$$\sum_{\substack{\text{for all } j\\ \text{not}\\ \text{containing } i}}\left[r_{ij}^2 + r_j^2\right] = 2^{2(m+1)} \qquad 4.23$$

Thus, combining the two equations:

$$\sum_{\substack{\text{for all } j\\ \text{not}\\ \text{containing } i}} r_{ij}^2 = 2^{m+1}.r_i \qquad 4.24$$

Thus it can be said that if all the coefficients in the original spectrum of the required function, as in equation 4.16, containing i in their subscripts are squared and then summed, compared with the coefficient r_i times 2^{m+1}, and found to be the same then the next state q_i can be obtained using an R-S bistable, with the S and R inputs equal to the J and K functions respectively, the spectra of which are given in

equation 4.19. If they are not the same, i.e. equation 4.24 is found to be invalid, then the suggested S and R functions are given by equation 4.25 which ensures that they are disjoint.

$$S = J.\overline{y}_i$$
$$R = K.y_i \qquad 4.25$$

In this instance, their spectral coefficients are shown to be:

$$\begin{aligned} S_0 &= \tfrac{1}{2}(r_i + r_0) + 2^m & R_0 &= \tfrac{1}{2}(r_i - r_0) + 2^m \\ S_j &= \tfrac{1}{2}(r_{ij} + r_j) & R_j &= \tfrac{1}{2}(r_{ij} - r_j) \\ S_i &= \tfrac{1}{2}(r_i + r_0) - 2^m & R_i &= \tfrac{1}{2}(r_0 - r_i) + 2^m \\ S_{ij} &= \tfrac{1}{2}(r_{ij} + r_j) & R_{ij} &= \tfrac{1}{2}(r_j - r_{ij}) \end{aligned} \qquad 4.26$$

for all subscripts j not including i.

T bistable

In this instance the function T can easily be obtained from its characteristic equation since the exclusive-or function is commutative. Thus:

$$T = q_i \oplus y_i \qquad 4.27$$

In the spectral domain this operation is equivalent to output spectral translation which was described in detail in section 2.2.1 in Chapter 2.

The spectra of each of the bistables can now be used in the hardware realisation where, for example, the J-K bistable may be chosen because of the fact that the J and K functions only require at most m variables since they are independent of the present state y_i. As another example, the T-type bistable may be useful since many of the

methods of hardware realisation involve spectral translations, and thus if the T-type bistable is used there could be a saving of one exclusive-or gate.

It has been said that a desirable feature of a serial input logic system is that it should consist of uniform logic hardware and that one such realisation that can be obtained uses universal logic modules. The following sections show that if these are chosen at the start of the design procedure then state assignment can be omitted altogether.

4.3 Modular Solution

4.3.1 Multiplexer-and Delay Module

The module that has been chosen is shown in Figure 4.9 which is a one-out-of-two multiplexer with a clocked delay.

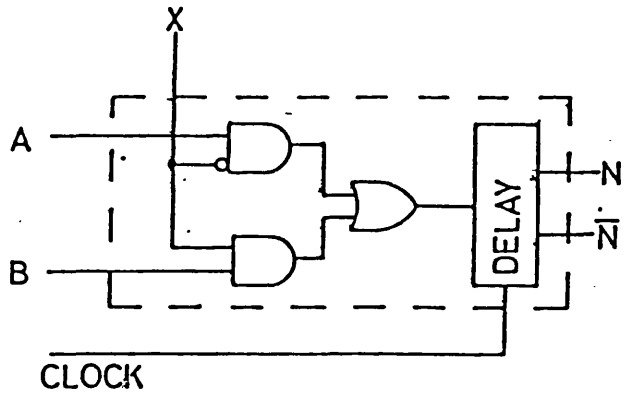


Figure 4.9 Universal logic module

The equation for N is given in equation 4.28.

$$N = \bar{x}.A + x.B \qquad 4.28$$

This module was chosen for three main reasons:

i) it has been shown [19,20] that any finite state machine can be represented by a network of these modules and that furthermore any "definite" sequential machine can be realised with the minimum delay Δ, i.e. one clock period. A definite machine is defined as one which does not require feedback [21], and it is clear from Figure 4.2, which is a valid realisation, that serial input logic systems are definite.

ii) it has been shown [22] that the maximum rate that data can arrive at a system input is equal to the reciprocal of the module's delay, and that this particular module has the minimum possible delay.

iii) it has been shown that a network of these modules, particularly if there is no feedback, is easily testable [23,24].

It would seem therefore that by these three facts alone the module is a good choice, particularly with regard to the speed aspect where, combining (i) and (ii), upon the arrival of the last input variable, or last bit of the input word, there is a delay which is equal to the delay of one module before the function output is given.

The design procedure using this module is as follows: consider the state diagram of Figure 4.6(b), and rename the states 1,2,3 and 4 as in Figure 4.10.

	x = 0	1	
S	N	N	Z
1	1	3	0
2	1	3	1
3	1	4	1
4	2	4	1

Figure 4.10 State table of the serial input sequential machine which realises the function $f(x) = x_1.x_2 + x_3$.

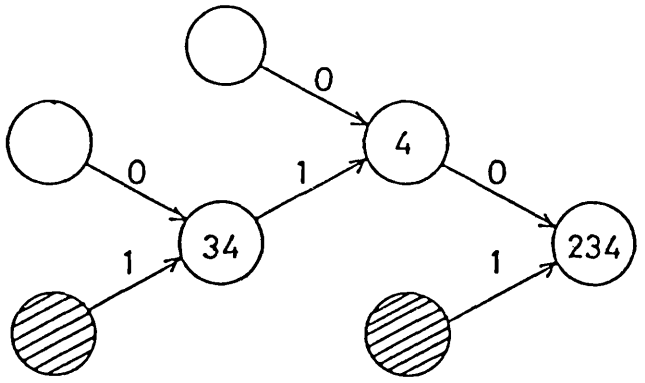
A graph is drawn which consists of interconnected nodes or circles in accordance with the following steps [7,19,20].

Step 1. Note all the states which have outputs of logic 1, which in this case are states 2, 3 and 4. These states are then grouped together and written in the first node as in Figure 4.11(a).

Step 2. All states which imply the previous states, i.e. have them as their next states N, are noted and two new nodes are drawn; one containing the states that imply the previous states when x = 0, and the other when x = 1. Arrowed lines are then drawn from each of these new nodes to the previous node, above which is written the corresponding x value.

Step 3. Step 2 is repeated for every new node until all nodes are terminal. A node is terminal if it contains no states or all the states in which case it it said to be empty or full and is shown as unshaded or shaded respectively. It can also be terminal if it contains a set of states, S_1 say, such that some other node also contains the set S_1, or if some other node contains the set, S_2 say, such that $S_1 \cup S_2 = S$, where S is the full set, in which case S_1 is said to be the inverse of S_2.

a)



b)

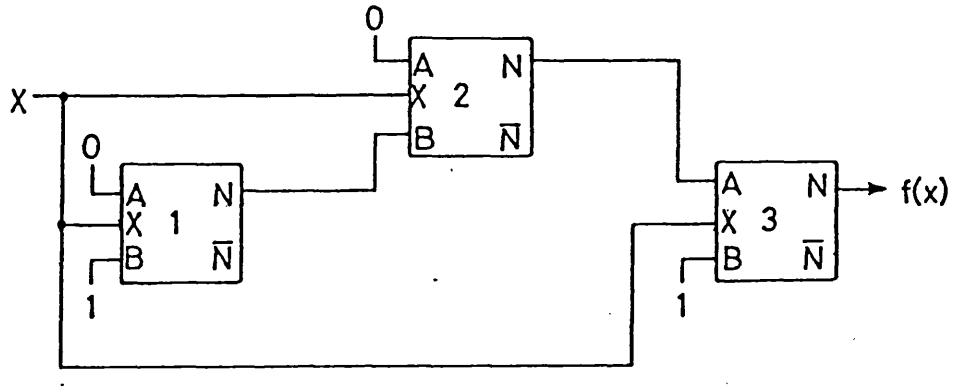


Figure 4.11 (a) Graph of nodes and branches
(b) Modular realisation

Figure 4.11(a) shows the graph for the example and Figure 4.11(b) shows the modular realisation.

Every non-terminal node in the graph is replaced by a module in the final network. The A and B inputs to each module correspond to the arrowed lines from previous nodes labelled 0 and 1 respectively and emerge from the N outputs of the modules corresponding to the implying nodes. If the implying node is empty or full then it is replaced by a constant logic 0 or 1 respectively. If a node is terminal because it contains the same or inverse set of states as another non-terminal node, then in the modular realisation the module corresponding to the non-terminal node supplies via its N or $\overline{N}$ output respectively the inputs of the modules implied by the terminal nodes.

Thus, a modular realisation is obtained directly from the state table and therefore does not require the state assignment stage of the design procedure.

4.3.2. Mode-Controlled Logic [1,2]

In section 4.1 it was stated that to date the only other approach to the realisation of a serial input system is mode-controlled logic, the modules which it uses being shown in Figure 4.12.

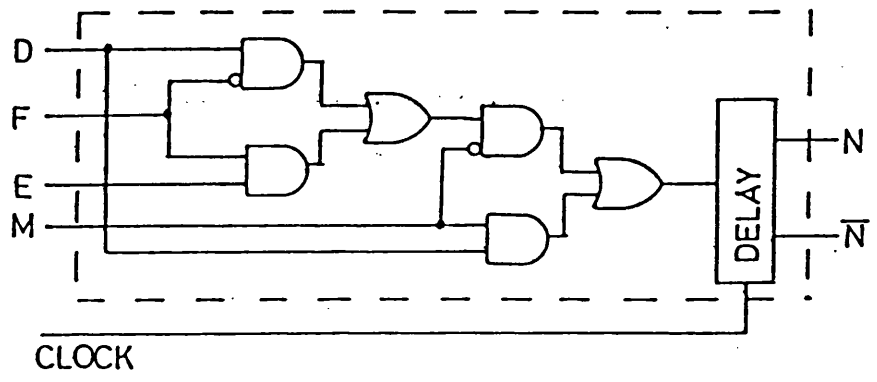


Figure 4.12 Mode-controlled logic module

Compared to the module in the previous section it can be seen that there is one extra input, M, the mode controller. The equation for N is now given as:

$$N = M.D + \overline{M}.(F.E + \overline{F}.D) \qquad 4.29$$

The mode is set so that either the variable D is simply copied at the output when M = 1 or a function of F and D is performed when M = 0. The function of F and D is preset by choosing a suitable value for E as in Table 4.6. Note that the latter case is identical to equation 4.28.

E	Function
0	AND
1	OR
$\overline{D}$	EX-OR

Assuming that not only F and D but their inverse also are available.

Table 4.6 Values of E for selected functions

Modules are then connected in tandem in a single path, i.e. every N output of a module goes to the D input of the module in front of it as in Figure 4.13.

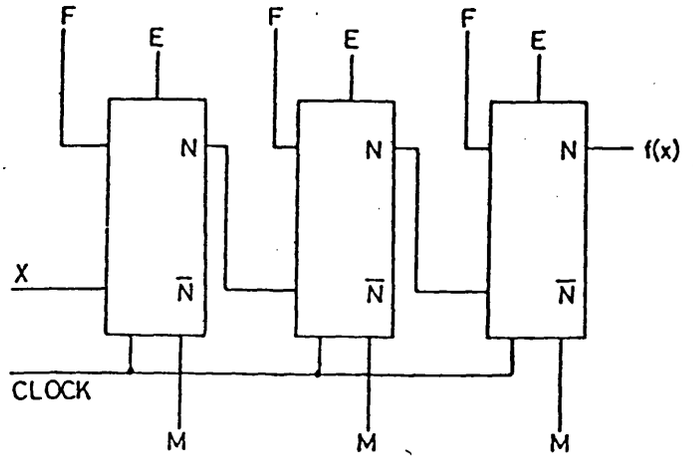


Figure 4.13 Tandem connected single path system of modules

The F input of every module is obtained as a feedback or feed-forward link from any other module. Thus the design procedure consists of finding the logical values for E and M and determining where to attach the inputs F. The approach is very unconventional and consists of firstly the use of spectral translation followed by an algorithm. The details of this are felt to be not required for this thesis since only a comparison of the solutions is necessary, but can be found in the listed references [1,2].

The main drawbacks of this system are:

i) additional circuitry is required for the mode controlling signals

ii) inherent latency in the system, i.e. the delay Δ between the last bit of the input word entering the system and the required output being generated is greater than one clock period

iii) undefined number of modules, i.e. no upper limit is given.

However, actual modules have been manufactured [3] using E.C.L. circuitry and have been operated at frequencies of up to 1 GHz. Typical systems that it has been applied to are, for example, serial input adders, multipliers, a threshold logic gate and a Grey code converter, details of which are all documented in the listed references [2,4].

4.4 Tree Structures

In section 4.2 the problem of utilising the don't care terms in the design procedure was confined to the process of state minimisation. However, two points must be raised:

i) state minimisation does not necessarily optimise the system since a reduction in the number of states does not guarantee a reduction in the number of state variables, i.e. the number of bistables, but merely introduces don't care terms into the next state function. For example, if originally a machine is specified by eight states, it requires three state variables to uniquely label each state. If, by the process of state reduction, the new number of states becomes five, three state variables are still required, only now there are three don't care states.

ii) a modular solution has been shown in section 4.3.1 which eases the design procedure by eliminating state assignment. Also, because of the strictly defined delays that each module involves, the timing of the system can be well organised and thus operate at a high speed. However, if it is decided at the outset that modules are to be used, as in the case of the mode-controlled logic system, then possibly there is a design procedure which can give a solution more directly, for example omitting the state reduction phase altogether. This could be done if the don't care terms in the initial state diagram could be incorporated into the graphical design method as in Figure 4.11(a). Fortunately this is possible and is therefore discussed in following sections.

4.4.1 Modular Solution Incorporation Don't Care Terms

A technique exists which gives a modular solution to incompletely specified machines [25], the modules being the same as that described in section 4.3.1. The technique uses next state mappings as in Figure 4.14 which shows the particular mappings for the next state

table of the function $f(x) = x_1.x_2 + x_3$ (reproduced from Figure 4.4(b)).

a)

	x = 0	1	
S	N	N	Z
0	2	3	0
1	2	3	1
2	4	4	-
3	4	5	-
4	0	1	-
5	1	1	-

b)

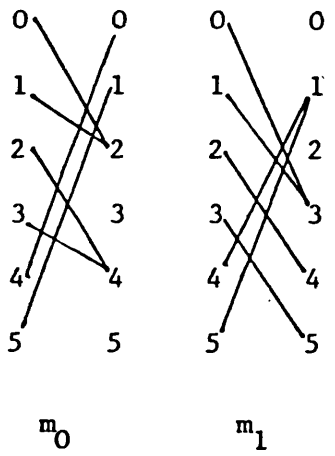


Figure 4.14 (a) State table for function $f(x) = x_1.x_2 + x_3$
(b) Next state mappings m_0 and m_1 where m_0 gives the next state when x = 0, and m_1 gives the next state when x = 1.

The mappings m_0 and m_1 correspond to the next states, on the right, of all the present states, on the left, for x equal to 0 and 1 respectively, e.g. the states 0 and 1 have the next state 2 when x is 0 as shown in the mapping m_0.

A simpler method than the mappings is to use equivalent next state matrices m_0 and m_1 as shown in Figure 4.15.

$$m_0 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad m_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.15 Next state matrices m_0 and m_1

Thus, if the columns of the state table of Figure 4.14(a) are regarded as column matrices, the product of the matrix m_0 with the column labelled S, i.e. the present states, results in a column matrix equivalent to N when x = 0, i.e. the next states, and the product of m_1 and S gives N when x = 1.

A reverse response tree can now be drawn where the initial node is drawn with a label equivalent to the column matrix Z in the state table transposed, i.e. converted to a row from a column. The descencant nodes are drawn and labelled with transposed column vectors equivalent to the product of m_0 and Z on the left and m_1 and Z on the right. This process is repeated for every node with the descendants' labels obtained from the products of the parent node's label and the matrices m_0 and m_1 as in Figure 4.16.

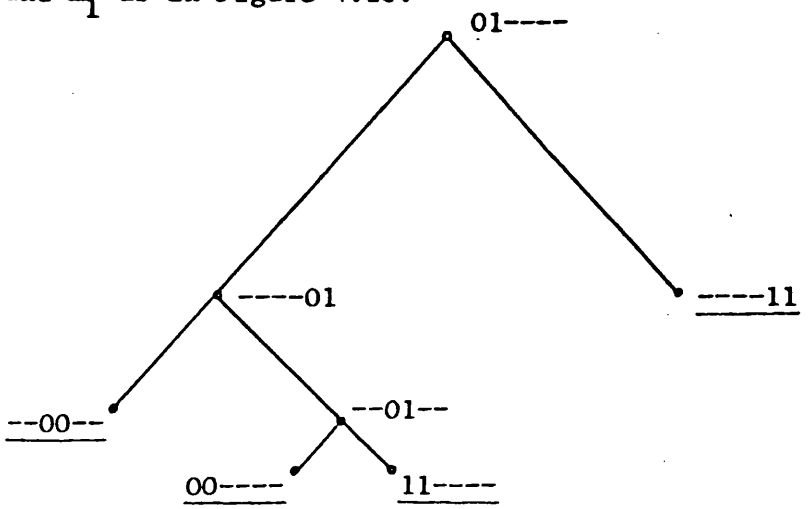


Figure 4.16 Reverse response tree for the state table of Figure 4.14(a)

The tree stops branching when all nodes are terminal, indicated by underlining the labels. Nodes become terminal, or are said to be pruned, if they are in accordance with any of the following conditions:

i) prune any node labelled L_1 if there exists another node labelled L_2 such that for every element of each label $L_1 = L_2$.

ii) as above but with $L_1 \neq L_2$, i.e. its inverse.

iii) prune any node labelled with all 0's or all 1's, or a mixture of 0's and don't cares or 1's and don't cares.

To obtain a modular network replace all non-terminal nodes with a module having its A and B inputs connected to the N outputs of the module equivalent of its left and right descendant nodes respectively (the A and B inputs are shown in Figure 4.9). If a descendant node is terminal because of condition (iii) then it is replaced by a constant 0 or 1, and if it is because of conditions (i) or (ii) then it is obtained from the N or $\overline{N}$ output of the module equivalent to the non-terminal node with the same or inverse label.

Clearly this is the same process as for a fully specified machine as described in section 4.3.1, and the resulting modular solution of the reverse response tree of Figure 4.16 is identical to that of Figure 4.11(b). However, it has been arrived at far more quickly and easily than when using the state reduction method.

4.4.2 General Solution

Reconsider the general three input variable function shown in Figure 4.7(a), (b) and (c). From the state table the next state matrices can be obtained and found to be as in Figure 4.17.

$$m_0 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \overline{p_0} & p_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \overline{p_2} & p_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ \overline{p_4} & p_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ \overline{p_6} & p_6 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad m_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \overline{p_1} & p_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \overline{p_3} & p_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ \overline{p_5} & p_5 & 0 & 0 & 0 & 0 & 0 & 0 \\ \overline{p_7} & p_7 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.17 Next state matrices

The elements of the next state matrices for any value of n are listed in Table 4.7.

i	m_0 j	m_0 m_{ij}	m_1 j	m_1 m_{ij}
1	3	1	4	1
2 to 2^{n-1}	2i - 2	1	2i	1
2^{n-1} + 1 to 2^n	1	$\overline{p}_{2i-2^n-2}$	1	$\overline{p}_{2i-2^n-1}$
2^{n-1} + 1 to 2^n	2	p_{2i-2^n-2}	2	p_{2i-2^n-1}
all others		0		0

Table 4.7 Elements m_{ij} of the next state matrices m_0 and m_1 where i and j represent the row and column numbers respectively

The next state matrices of Figure 4.17 can be used to generate the labels on the nodes of a reverse response tree using the same method as in section 4.4.1, the initial node being labelled with the transpose of the column matrix equivalent to Z in Figure 4.7(c). The result is shown in Figure 4.18.

It can be seen from this reverse response tree that all the nodes on the fourth level, i.e. after three branchings, can be terminated or pruned in accordance with the pruning conditions stated in the previous section, i.e. labels consisting of 0's and don't cares or 1's and don't cares. Therefore the maximum number of nodes and hence modules is $2^n - 1$. However, the nodes on the third level consist of don't care terms and two p_i terms so that every node has the possibility of having one of the following four labels:

i) $--00----$

ii) $--01----$

iii) $--10----$

iv) $--11----$

If any of the nodes have the labels (i) or (iv) then they can be pruned in the same manner as the fourth level nodes. Of the remaining nodes, let one have the label (ii); then any other nodes with labels (ii) or (iii) can be pruned by the first and second conditions of

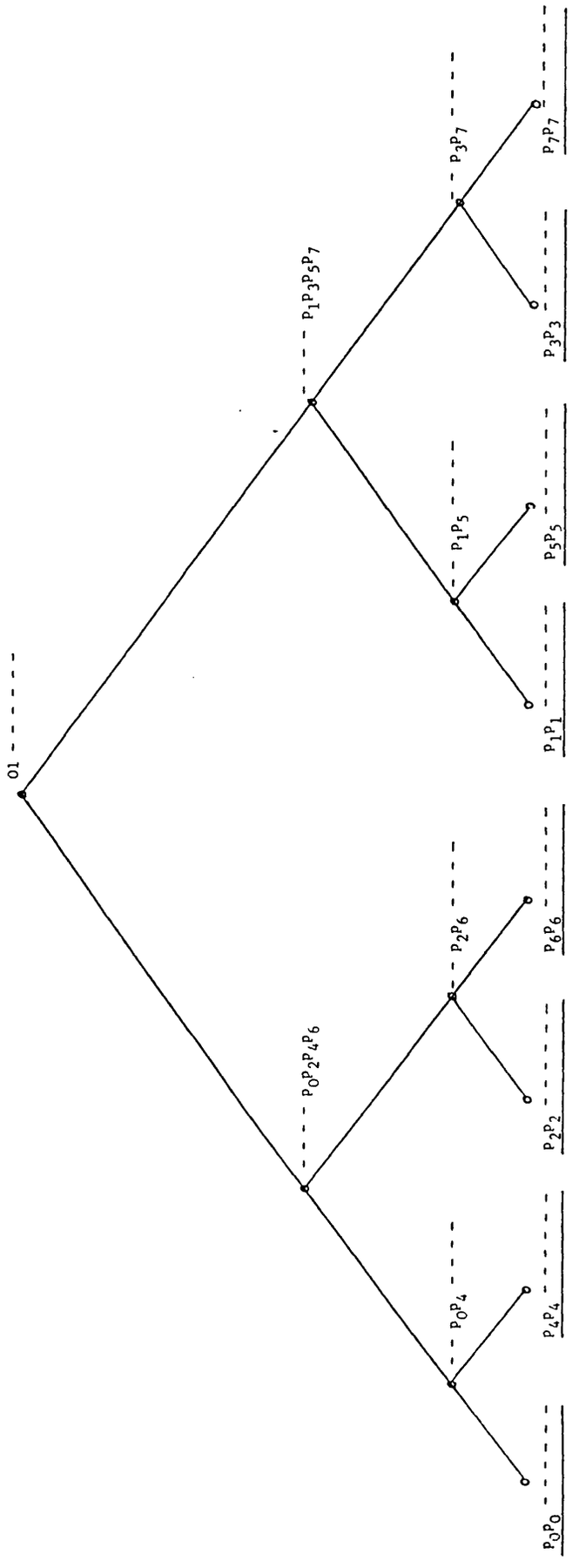


FIGURE 4.18 GENERAL REVERSE RESPONSE TREE FOR n = 3

pruning respectively, i.e. because the label appears elsewhere in the tree or that the inverse appears elsewhere. Therefore there can only be one unpruned node on the third level so that the maximum number of modules in this case is four, or more generally 2^{n-1}. In fact even this figure is a little too high because using the same argument as above it can be shown that the maximum number of modules on any level, k say, is given in equation 4.30.

$$\text{Maximum number of modules on the } k^{th} \text{ level} = 2^{2^{n-k+1}-1} - 1 \qquad 4.30$$

Thus, for example, when n = 5, the number of modules on the fourth level would initially seem to be at most eight, but by applying equation 4.30 with n = 5, k = 4 it is found to be seven. Thus the maximum number of modules instead of being $2^{5-1} = 16$ is 15. The maximum number of modules for any function is given in equation 4.31.

$$\text{Maximum number of modules} = \sum_{k=1}^{n+1} \min.(2^{k-1}, 2^{2^{n-k+1}-1} - 1) \qquad 4.31$$

This equation takes the sum of all the lower of the two expressions in the brackets for all values of k.

So far no account has been taken of the don't care terms which can be assigned values in order to prune even more nodes. Three possible methods of assigning values will now be considered and the pros and cons of each discussed.

4.4.2.1 Random Assignment

Labels on the same level in Figure 4.18 can be pruned if their corresponding p_i values are all the same or all different in accordance with the conditions for pruning. However, labels on nodes on different levels are such that the p_i terms in one label correspond in position

with don't care terms in the other or vice versa. Therefore any node on any level can be pruned by setting its don't care terms so that its label corresponds to any other label. This would appear to be an advantage initially, but on closer examination it is found that there are a number of drawbacks due to the fact that labels are obtained from the labels of parent nodes using the next state matrices. Thus, if a don't care term is assigned a value then some of the don't care terms in the descendant nodes have to be assigned the same value. Nodes which have already been pruned may now have their labels changed which may make them unable to be pruned and hence a great deal of confusion arises. Also, the nodes on the fourth level which previously could all be pruned may now not be able to be pruned by the previous argument, which therefore means that the reverse response tree may not have a finite number of levels so that the maximum number of modules cannot be determined as before. As an example of this, consider what happens if it is decided to assign don't care values such that the following nodes are the same:

$$---- P_0P_2P_4P_6 = -- P_5P_7 ---- \text{ i.e. } -- P_5P_7P_0P_2P_4P_6$$

The resulting reverse response tree is shown in Figure 4.19 where not all the sub-trees are drawn for ease of identification. However, the one node which is fully expanded shows that the number of levels is doubled and the number of modules greatly increased. Since the purpose of assigning values to the don't care terms is to reduce the size of the tree, it can be seen that this method is not particularly good.

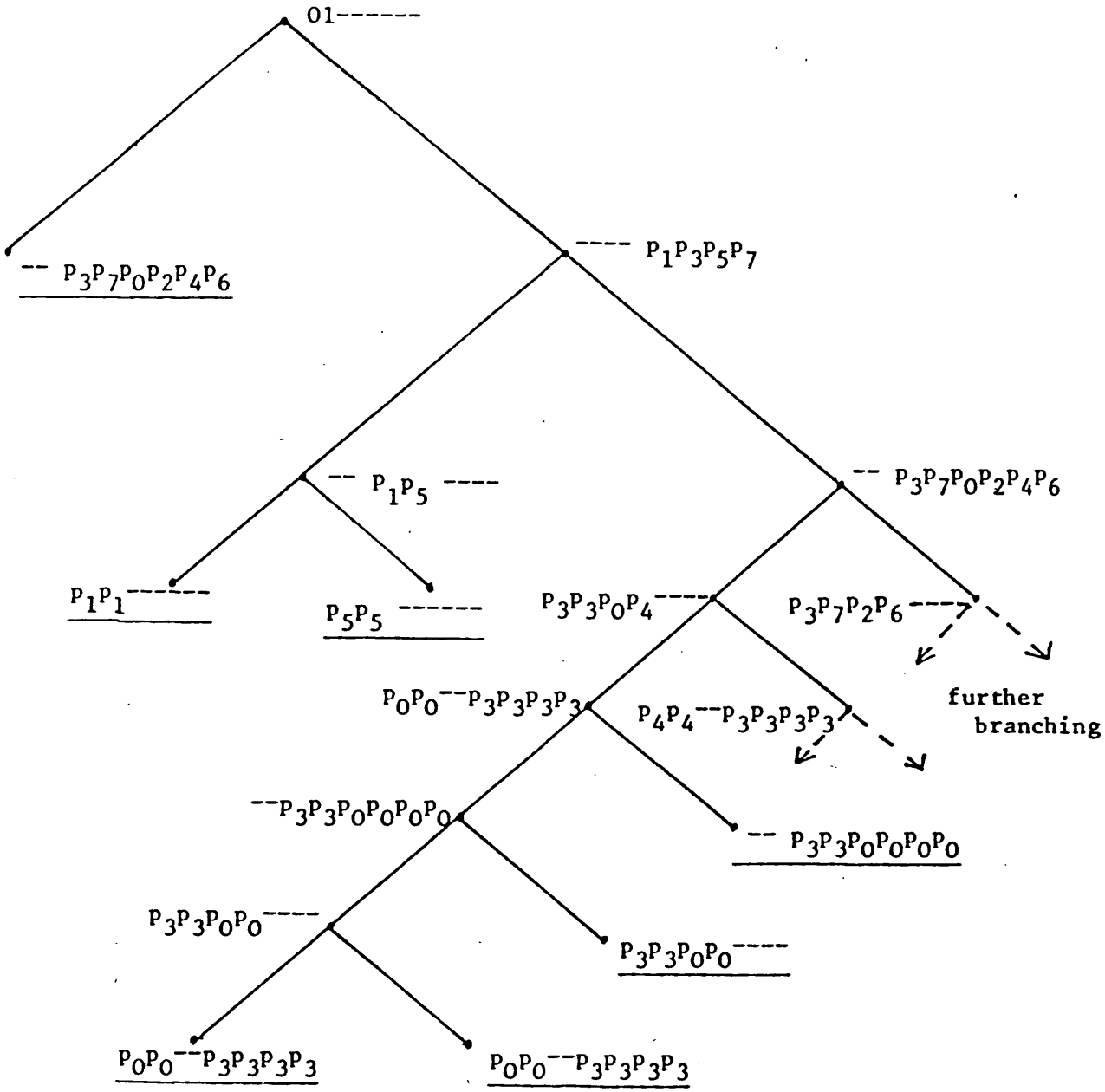


Figure 4.19 Randomly assigned don't care terms in the reverse response tree

4.4.2.2 Invisible Don't Cares [26]

If only nodes on the same level are compared for pruning then the don't care terms are superfluous and can be ignored. This results in the reverse response tree of Figure 4.20 which is a much simpler one than before.

Consider now a similar tree with the initial node labelled with the function f(x) that is being realised. The labels on the second level nodes are obtained by setting x_3 to 0 for the left hand node and x_3 to 1 for the right hand node. Further descendant nodes are labelled by setting x_2 and finally x_1 to 0 and 1 as in Figure 4.21 where the specific function $f(x) = x_1.x_2 + x_3$ is shown. Nodes are pruned if they have labels consisting of 0's, 1's or Boolean expressions which are the same or the inverse of some other nodes on the same level.

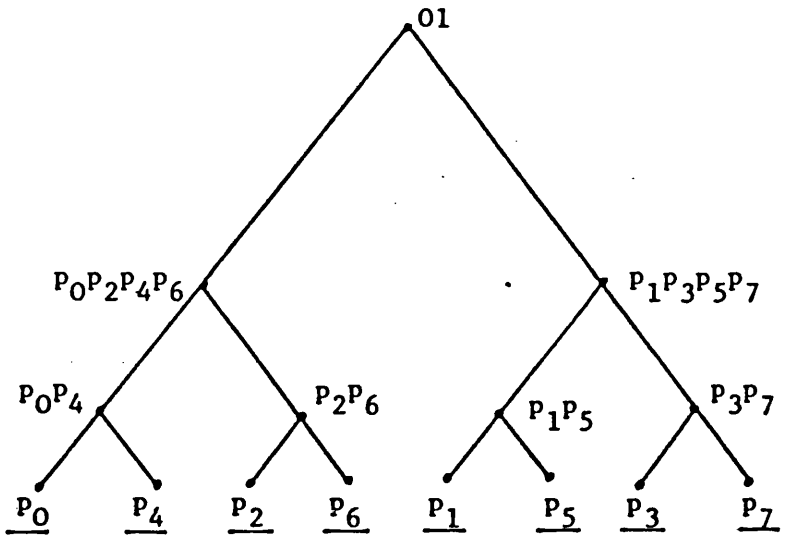


Figure 4.20 Reverse response tree without the don't care terms

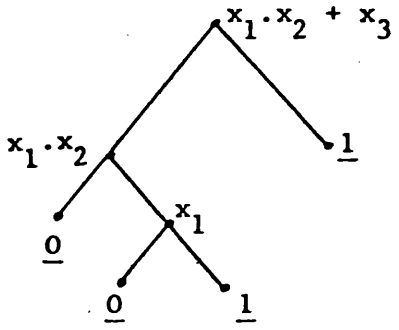


Figure 4.21 Reverse response tree of function $f(x) = x_1.x_2 + x_3$

Comparing these two reverse response trees shows that in fact they are identical. For example, consider the situation where the two nodes labelled p_0p_4 and p_1p_5 are the same and therefore one of them is equivalent to the initial Boolean expression with $x_3 = x_2 = 0$ for the latter and $\bar{x}_3 = x_2 = 0$ for the former. Thus the two labels are:

$$\bar{x}_1p_0 + x_1p_4 \quad \text{and} \quad \bar{x}_1p_1 + x_1p_5 \qquad 4.32$$

Therefore , for the two expressions to be identical, it is required that $p_0 = p_1$ and $p_4 = p_5$, which is then as for the other reverse response tree.

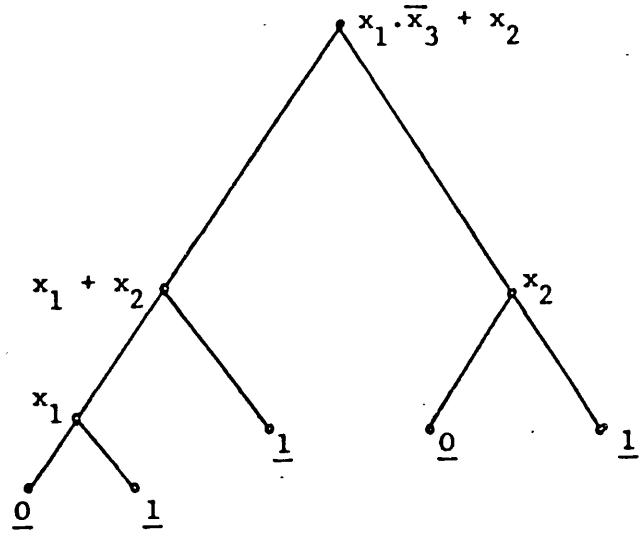
Comparing the reverse response tree of Figure 4.21 with that of Figure 4.16, it is found that they are identical and would result in the same modular realisation. It is clear, however, that the derivation of the former is far simpler than the latter which relied on state minimisation as its means of utilising the don't care terms. However, this method too has one drawback which is that only nodes on the same level can be pruned if they have the same or inverse labels. As an example of a situation where this drawback becomes apparent consider the function given in equation 4.33.

$$f(x) = x_1.\bar{x}_3 + x_2 \qquad 4.33$$

The reverse response tree and modular realisation is shown in Figure 4.22(a) and (b).

The mode in which the system operates is that as an input variable arrives at the system it is processed by the modules in one particular time slot. For example, the input variable x_2 would be processed by the modules 2 and 3 in the time slot 2, the result of which would then pass on to module 4 where further processing takes place with the next

a)



b)

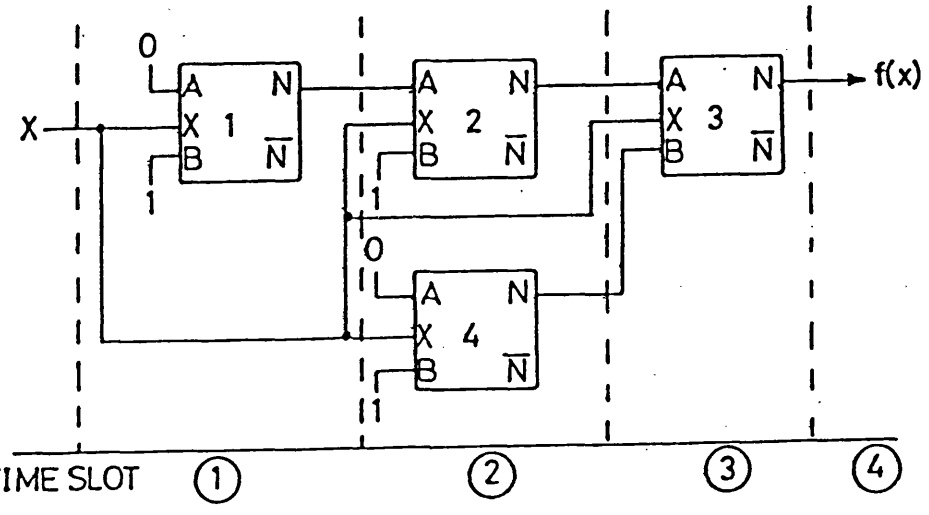


Figure 4.22 (a) Reverse response tree
(b) Modular realisation

input variable x_3 in time slot 3. Therefore, modules in a particular time slot are only required to operate in that time slot, at all other times they are redundant. Clearly, however, the modules are still operating and it is only the results of these operations that are redundant. Thus it can be seen in Figure 4.22 that module 1 processes x_1 and module 3 processes x_2 one clock period later, and since both receive the same input information it must be the same process so that in fact modules 1 and 3 are identical and interchangeable, resulting in the simplified network of Figure 4.23.

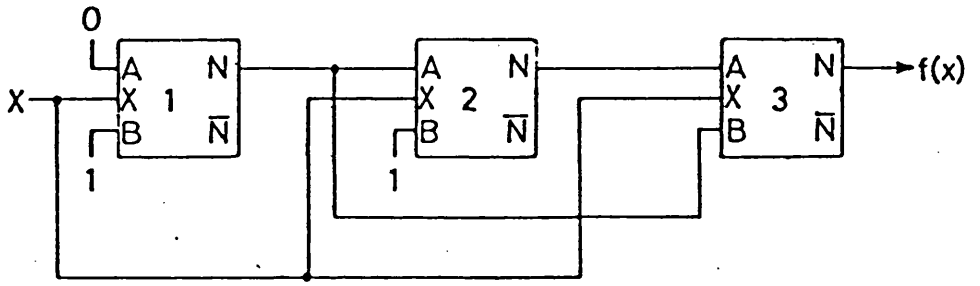


Figure 4.23 Improved modular realisation of $f(x) = x_1.\overline{x_3} + x_2$

It is difficult to discover which modules are identical and interchangeable from this method and so the following improved method which allows pruning of nodes on different levels with the same or inverse labels has been developed.

4.4.2.3 Specific Assignment of Don't Care Terms

In this method the labels on the nodes of the reverse response tree are first obtained as in Figure 4.18 and then adjusted in the following manner: a label which has k specified elements has its don't care terms to the right of these k elements assigned to the k elements

in sets of k. Thus the label;

k assigned elements — Two groups of k unassigned elements

$$- - - \ldots - L_1 L_2 \ldots L_k - - - \ldots - - - - \ldots -$$

becomes — All three groups of k elements are assigned with L_1 to L_k

$$- - - \ldots - L_1 L_2 \ldots L_k L_1 L_2 L_3 \ldots L_k L_1 L_2 L_3 \ldots L_k$$

The reverse response tree for a general function of n = 3 now becomes as in Figure 4.24.

The initial node is left as it is since it plays no part in the pruning due to the fact that no node can have the same label as one of its descendant nodes. This is because, as stated at the beginning of the chapter, a serial input logic system is a definite one, i.e. needs no feedback loops. In the case of this particular labelling system it is impossible for a descendant node to be pruned by virtue of having the same label as a parent node, e.g. let the label $- - p_0 p_4 p_0 p_4 p_0 p_4$ be the same as the label $- - - - p_0 p_2 p_4 p_6$. For this to happen it is required that $p_0 = p_2 = p_4 = p_6$ in which case the latter would have already been pruned by virtue of the fact that it can be replaced by a logical constant 0 or 1. However, labels on different levels which are not descendants of each other can be the same and therefore pruned. As an example consider the two nodes labelled $- - - - p_0 p_2 p_4 p_6$ and $- - p_1 p_5 p_1 p_5 p_1 p_5$. In order for these two labels to be the same, it is required that equation 4.34 is obeyed.

$$p_1 = p_0 = p_4 \qquad 4.34$$

$$p_5 = p_2 = p_6$$

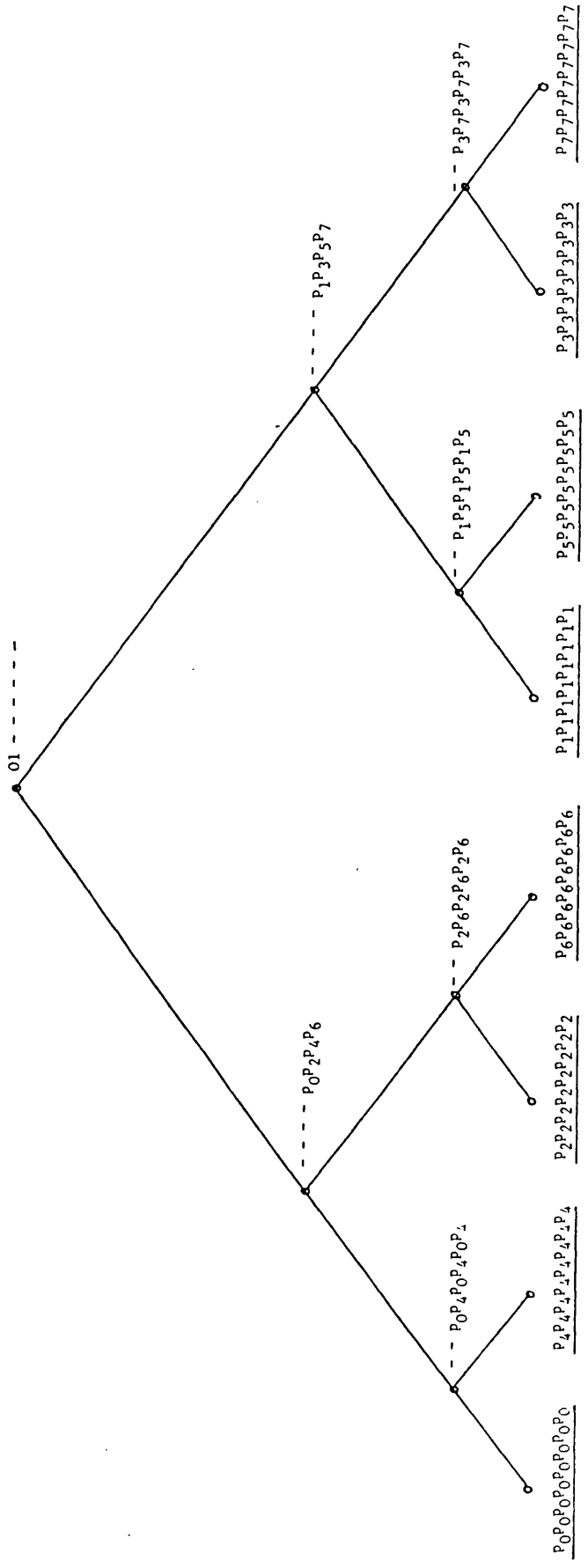


FIGURE 4.24 REVERSE RESPONSE TREE WITH REPEATEDLY ASSIGNED DON'T CARE TERMS

Now if, as in the case of the invisible don't cares of section 4.4.2.2, a similar reverse response tree is drawn with the initial label being the Boolean expression of the required function, and descendant nodes labelled by the corresponding decomposition about the appropriate input variables, then the Boolean expressions for the two nodes under examination would be as in equation 4.35.

$$\bar{x}_1 \cdot \bar{x}_2 \cdot p_0 + \bar{x}_1 \cdot x_2 \cdot p_2 + x_1 \cdot \bar{x}_2 \cdot p_4 + x_1 \cdot x_2 \cdot p_6 = \bar{x}_1 p_1 + x_1 p_5 \qquad 4.35$$

This equation would be satisfied if equation 4.36 is satisfied, which is not the required condition.

$$p_0 = p_2 = p_1 \qquad 4.36$$

$$p_4 = p_6 = p_5$$

If, however, the input variables in the Boolean expressions for labels are adjusted in accordance with the following rule, the correct conditions can be met.

Rule 4.1

If the input variable arriving at the system is x_i, then all input variables x_j must be altered to x_{n-i+j} and the labels of the descendant nodes obtained by decomposing about the variable x_n. Thus, on the first level, the incoming variable is x_n, so that $i = n$ and all variables remain unchanged. On the second level, the incoming variable is x_{n-1}, so that $i = n - 1$ and all terms x_j on that level become x_{j+1}. Thus, in general when drawing the reverse response tree, as each level is reached, the input variables in the Boolean expressions have their subscripts incremented.

Using this labelling scheme, the required equation to be satisfied in order that the two nodes have the same label is given by equation 4.37.

$$\bar{x}_2 \cdot \bar{x}_3 \cdot p_0 + \bar{x}_2 \cdot x_3 \cdot p_2 + x_2 \cdot \bar{x}_3 \cdot p_4 + x_2 \cdot x_3 \cdot p_6 = \bar{x}_3 \cdot p_1 + x_3 \cdot p_5 \qquad 4.37$$

Now the conditions for this equation to be satisfied are the same as in equation 4.34, so that this labelling scheme ensures that nodes with the same or inverse Boolean expressions can be pruned regardless of what level they are on.

Applying this method to the example given in equation 4.33, the reverse response trees of Figure 4.25 can be drawn.

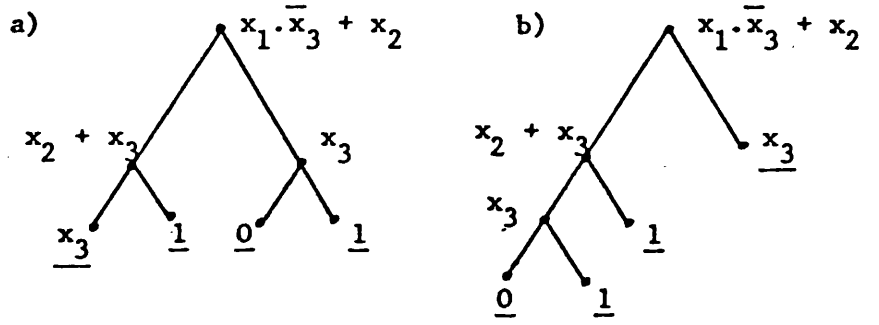


Figure 4.25 (a) and (b) Reverse response trees of function $f(x) = x_1 \cdot x_3 + x_2$

Both trees in Figure 4.25 are the same in that the modular realisation of them would be identical to Figure 4.23. It can therefore be seen that this method is simple to apply and ensures that the number of modules in the solution is kept to a minimum. The procedure can be summarised into the following steps:

Step 1 Express the required function as a Boolean expression and label the initial node with it.

Step 2 Decompose this expression about the variable x_n, labelling the left hand descendant node with the expression when $x_n = 0$, and the right hand descendant node with the expression when $x_n = 1$. Increment the subscripts of all the input variables in the descendant nodes' expressions.

Step 3 Repeat step 2 another n - 1 times, pruning all nodes labelled 0,1, or those that have the same or inverse labels as the labels of any nodes that have appeared in the tree.

Step 4 Convert to the modular realisation by replacing all non-terminal nodes with modules and wire in the same manner as described previously in section 4.4.1.

This reverse response tree with Boolean expressions for labels gives rise to a very interesting result. Consider again the reduced machine consisting of the implied maximal compatibles from section 4.2.2. Figure 4.26 shows the state table, next state matrices, and reverse response tree for a general function of $n_v = 3$ obtained from the general maximal compatibles of Table 4.4 and the flow diagram of Figure 4.7(a).

The reverse response tree is of particular interest because it has the same structure as all the others obtained including the feature that it terminates after three branches. However, when compared with the reverse response tree of Figure 4.24 it is found to be almost identical, the only difference being the fact that it is fully specified. The reverse response tree with Boolean expressions for labels is found to be also interchangeable with this new reverse response tree, and since it has been found to be sufficient to identify all nodes that can be pruned and therefore optimal, it can be said that the maximal compatibles machine is also optimal, at least when a modular solution is desired. Thus the arguments presented at the beginning of this section against the state reduction method for producing the minimal solution do not apply in this instance and therefore the reverse response tree with Boolean expressions or that of Figure 4.26(c) can be used to obtain a modular solution.

a)

x = 0 , 1

S	N	N	Z
p_0^{24}	p_0^{24}	p_1^{35}	p_0
p_1^{35}	p_2^{26}	p_3^{37}	p_1
p_2^{26}	p_4^{24}	p_3^{37}	p_2
p_3^{37}	p_6^{26}	p_7^{37}	p_3
p_4^{24}	p_0^{24}	p_1^{35}	p_4
p_5^{35}	p_2^{26}	p_3^{37}	p_5
p_6^{26}	p_4^{24}	p_5^{35}	p_6
p_7^{37}	p_6^{26}	p_7^{37}	p_7

b)

$$m_0 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \end{bmatrix} \qquad m_1 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

Figure 4.26 (a) General reduced machine state table
(b) Next state matrices

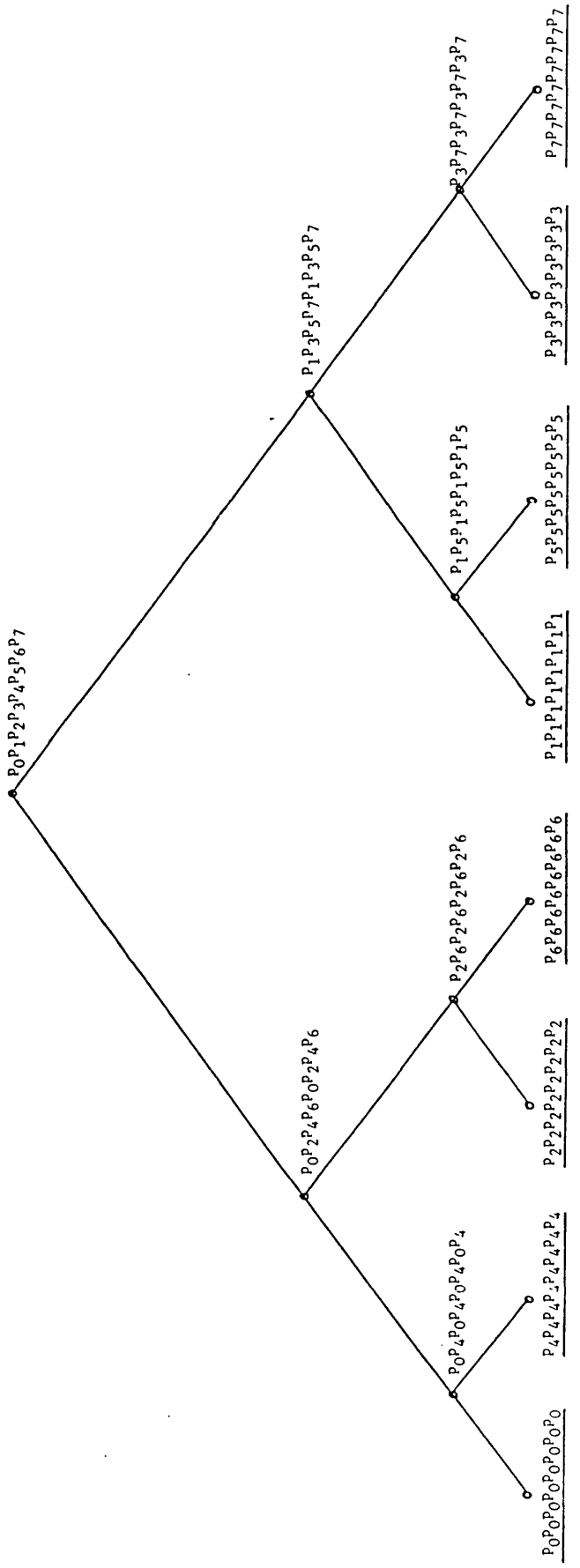


FIGURE 4.26(c) REVERSE RESPONSE TREE

If the reverse response tree with the minterm labels rather than the Boolean expressions is used then the next state matrices are used to obtain each label. Table 4.8 summarises the elements of the matrices for any value of n.

i	m_0 j	m_{ij}	m_1 j	m_{ij}
1 to 2^{n-1}	2i - 1	1	2i	1
2^{n-1} + 1 to 2^n	2i - 1 - 2^n	1	2i - 2^n	1
All others		0		0

Table 4.8 Summary of elements of next state matrices

4.4.3 Additional Features

So far it has been shown that given any function f(x) which is to be realised in serial form, the optimal solution can be obtained by finding a machine consisting of the implied maximal compatibles and then choosing any appropriate type of logic elements. If the realisation is to be modular then this too could be obtained from this machine, but it has been shown that there is a more direct approach using a reverse response tree with the Boolean expressions of the function and its decomposition about the appropriate input variables as the labels on the nodes. However, it has also been shown that this is equivalent to the reverse response tree of the machine consisting of the maximal compatibles which can also be obtained directly by the use of the next state matrices. It is this latter reverse response tree which proves useful when dealing with incompletely specified functions, since the don't care terms can be present in the node labels which is not possible when Boolean expressions are used.

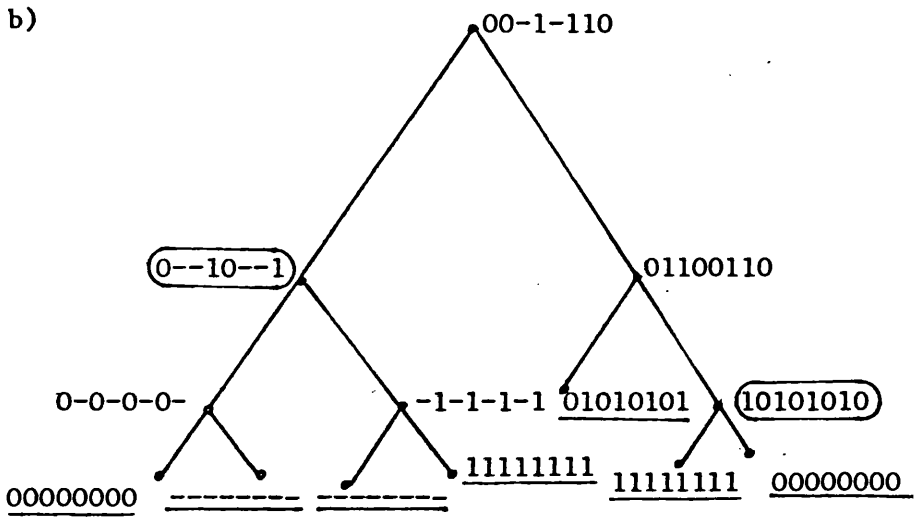
4.4.3.1 Incompletely Specified Functions

Given a function f(x) which is incompletely specified, it is possible to utilise the don't care terms to reduce the number of modules required to realise it. As an example, consider the function shown in the truth table of Figure 4.27(a).

a)

$x_1x_2x_3$	f(x)
0 0 0	0
0 0 1	0
0 1 0	-
0 1 1	1
1 0 0	-
1 0 1	1
1 1 0	1
1 1 1	0

b)



c)

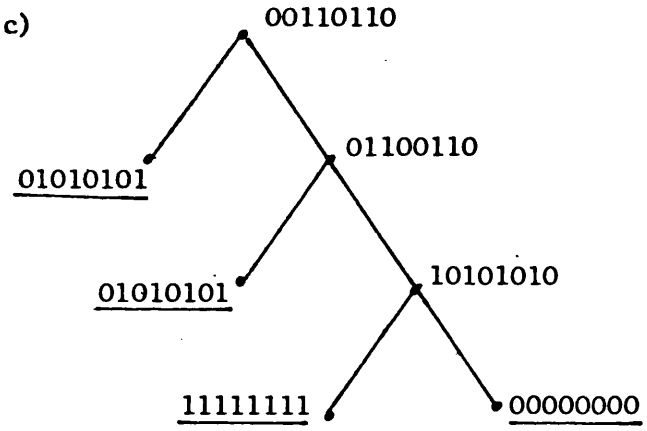


Figure 4.27 (a) Truth table of required function

(b) Initial reverse response tree with don't cares

(c) Reverse response tree after the assignment of the don't care values.

Figure 4.27(b) shows the initial reverse response tree and the circled labels are the ones selected such that one can be pruned by virtue of having its label the inverse of the other. Special attention must be drawn to the fact that nodes cannot be pruned if they have the same or inverse labels of one of their descendant nodes since this would violate the feedback condition, i.e. no feedback permitted.

4.4.3.2 Multi-Output System

If more than one function is required to occur simultaneously in a system, then a reduced system can be obtained by drawing the reverse response trees for each output function and then cross-pruning between them. As an example of this, consider the situation of a binary coded decimal (B.C.D.) to Grey code converter [9], the truth table for a four bit converter being shown in Figure 4.28.

Since there are don't care terms in the functions the reverse response trees with actual minterm values are used. The four trees for G_1, G_2, G_4 and G_8 are shown in Figure 4.29, where it can be seen that each of the initial labels of the three latter outputs can be found in the tree for G_1 and therefore can be pruned. The modular realisation is shown in Figure 4.29(e).

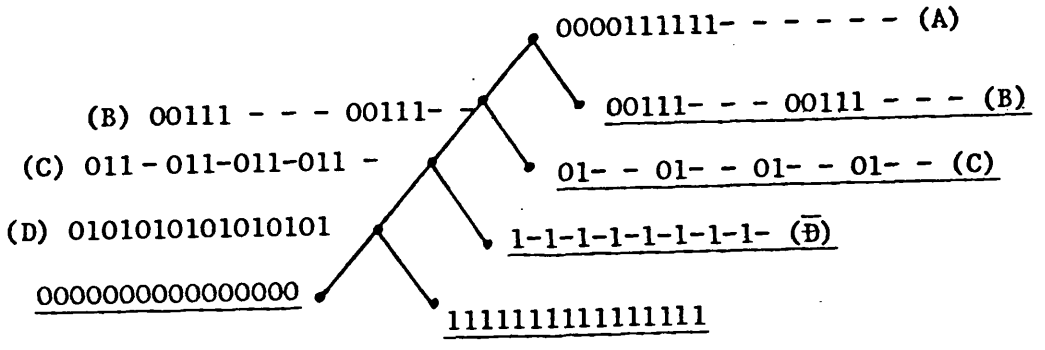
The letters in brackets next to the labels in the reverse response trees are included for ease of identification of nodes with equivalent labels.

It can be shown that if the number of bits in the conversion is n, then the number of modules is n, the additional modules being placed after module 4 and wired in the same manner as module 4 is to module 3.

$B_1B_2B_4B_8$	$G_1G_2G_4G_8$
0 0 0 0	0 0 0 0
0 0 0 1	0 0 1 1
0 0 1 0	0 1 1 0
0 0 1 1	0 1 0 1
0 1 0 0	1 1 0 0
0 1 0 1	1 1 1 1
0 1 1 0	1 0 1 0
0 1 1 1	1 0 0 1
1 0 0 0	1 0 0 0
1 0 0 1	1 0 1 1
1 0 1 0	- - - -
1 0 1 1	- - - -
1 1 0 0	- - - -
1 1 1 0	- - - -
1 1 1 1	- - - -

Figure 4.28 Four bit B.C.D. to Grey code converter

a)



b) • 0011110000 - - - - - (B)

c) • 0110011001 - - - - - (C)

d) • 0101010101 - - - - - (D)

e)

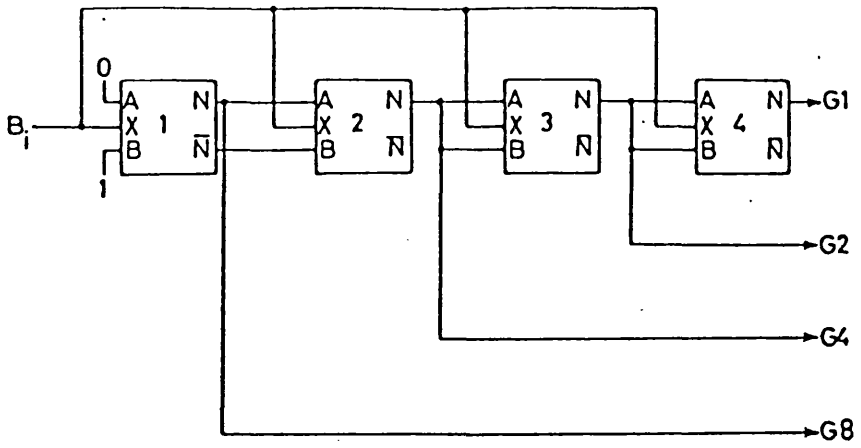


Figure 4.29 (a), (b), (c) and (d) Reverse response trees for G_1, G_2, G_4 and G_8

(e) Modular realisation

4.4.3.3 Multi-Input Systems

It may be necessary in some situations to be able to process more than one word formatted serial input in a system, such as in arithmetic functions. The design in this situation would also involve tree structures as before but each node would branch to 2^m descendant nodes, where m is the number of serial inputs to the system. The hardware realisation would then consist of modules having 1-out-of-2^m multiplexers and delays.

4.4.3.4 Serial Multi-Outputs

So far the output of the system has been considered as an n bit word, only one bit of which is relevant, the remainder being don't care terms as in Figure 4.1. The multi-output system that has been described in section 4.4.3.2 had its outputs in parallel, each being an n bit word. However, it may be desirable to have only one output, each bit at least more than one of its bits, being a different function as in Figure 4.30.

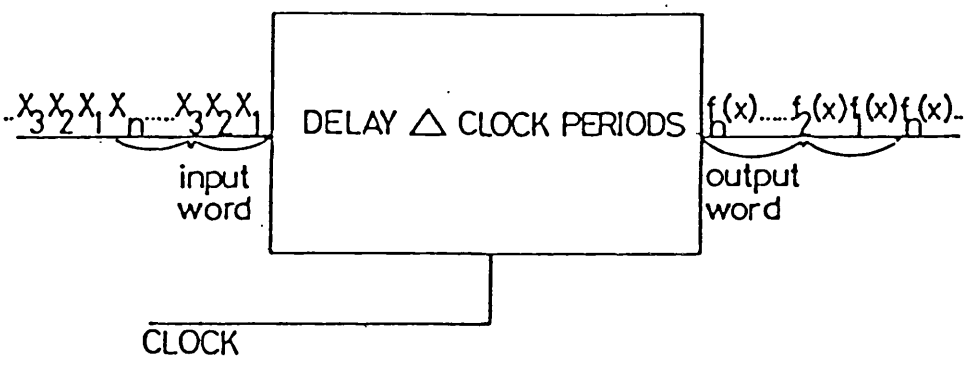


Figure 4.30 Schematic diagram of serial multi-output system

If all the functions in the output word are specified then the modular solution can be found by applying the techniques described in section 4.3.1. However, an important point to note is that if the delay Δ is to be kept at its minimum value of one, then each of the functions f_i must only be a function of the first i variables, i.e. x_1 to x_i, otherwise it would be a function of the variables from more than one input word. If they are functions of more than i variables and an uncorrupted signal is required then the delay Δ has to be increased, i.e. latency would have to be introduced into the system but the design procedure would remain the same.

If some of the functions are not specified then the suggested approach is to use the method described in section 4.4.1, namely to draw a flow diagram, state table, next state matrices, and finally reverse response tree. This method cannot be simplified as before since the conditions are much different.

As examples of the fully specified functions consider the Grey code converter described in Figure 4.28 and then a serial full adder circuit.

Example 1

On examination of the truth table of Figure 4.28 it can be found that assignment of the don't care values could give functions which are dependant on a reduced number of variables, the equations being given in equation 4.38.

$$G_1 = B_1 \oplus B_2 \qquad 4.38$$

$$G_2 = B_2 \oplus B_4$$

$$G_4 = B_4 \oplus B_8$$

$$G_8 = B_8$$

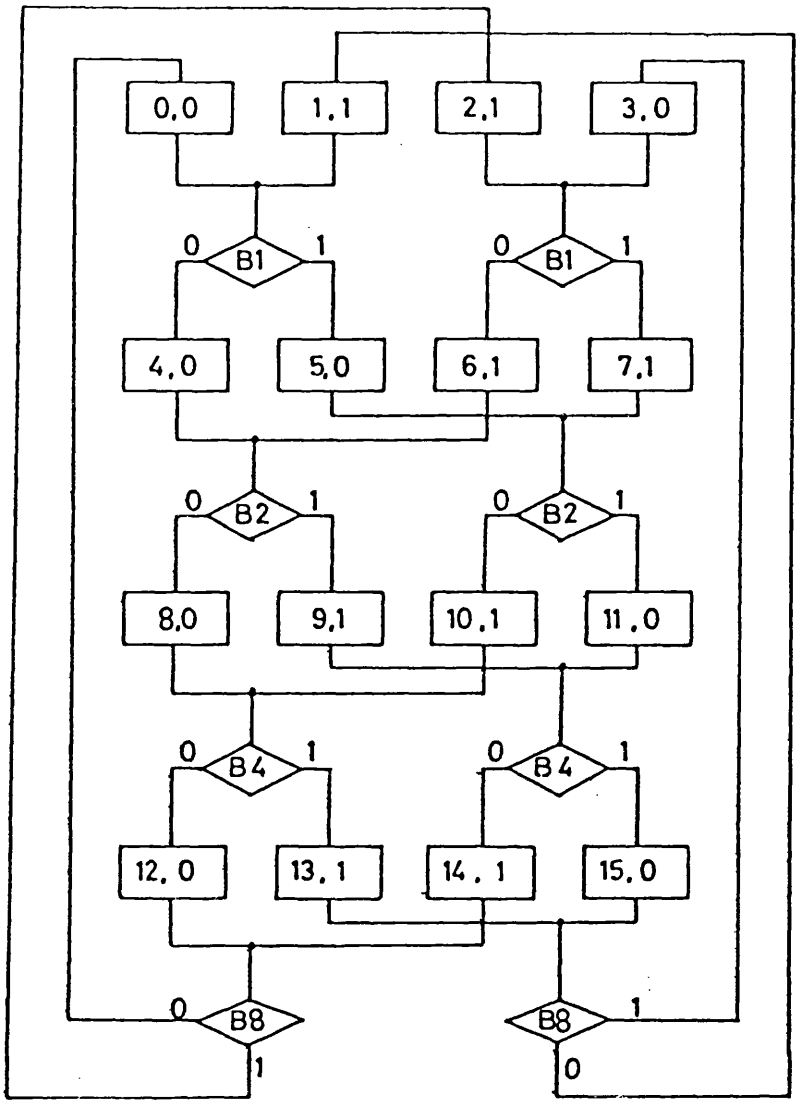
A flow diagram of a system which would realise these functions as a single output serial stream is shown in Figure 4.31(a). If the methods for state reduction are applied it is found that the flow diagram cannot be reduced. Applying the technique for obtaining the modular solution results in the network of Figure 4.31(b).

The system contains modules 1 to 4 which provides a cyclic sequence of 0's and 1's that serve to invert or not invert the incoming input variable respectively. In fact the sequence is 1110, and had there been n bits to convert,the sequence would have consisted of n - 1 logic 1's followed by a logic 0. When, for example, B_1 arrives at the input, module 6 simply transfers B_1 to the input of module 7, whereas module 5 transfers the inverse, i.e. $\overline{B}_1$. Thus, when the next input B_2 arrives at module 7, it is processed so that the output of module 7 is as in equation 4.39.

$$\text{output} = B_1.\overline{B}_2 + \overline{B}_1.B_2 = B_1 \oplus B_2 = G_1 \qquad 4.39$$

However, when the input B_8 arrives at the system, both module 5 and 6 transfer B_8 to the inputs of module 7 since this time the logical

a)



b)

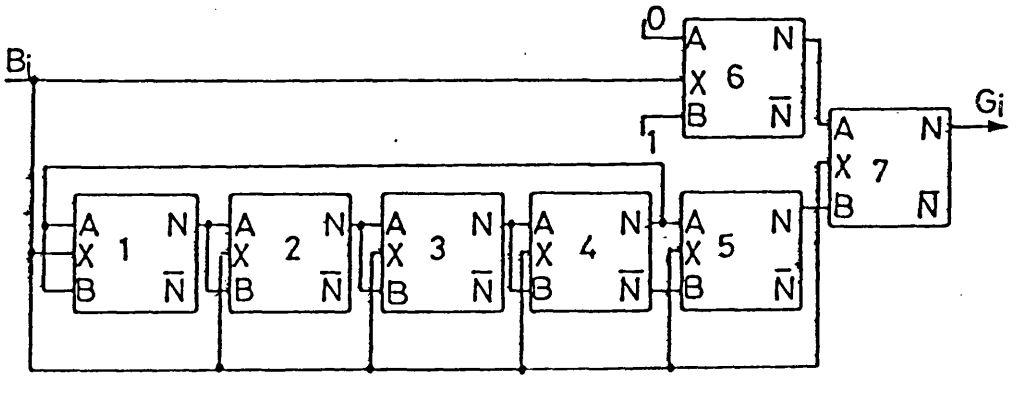


Figure 4.31 (a) Flow diagram for system which realises a four bit B.C.D. to Grey code converter

(b) Modular realisation

constant from module 4 is 0. Thus upon the arrival of the next input, which is B_1 of the next input word, the function given by equation 4.40 is produced at the output, which is the required one.

$$\text{output} = B_8.\overline{B}_1 + B_8.B_1 = B_8 = G_8 \qquad 4.40$$

Example 2

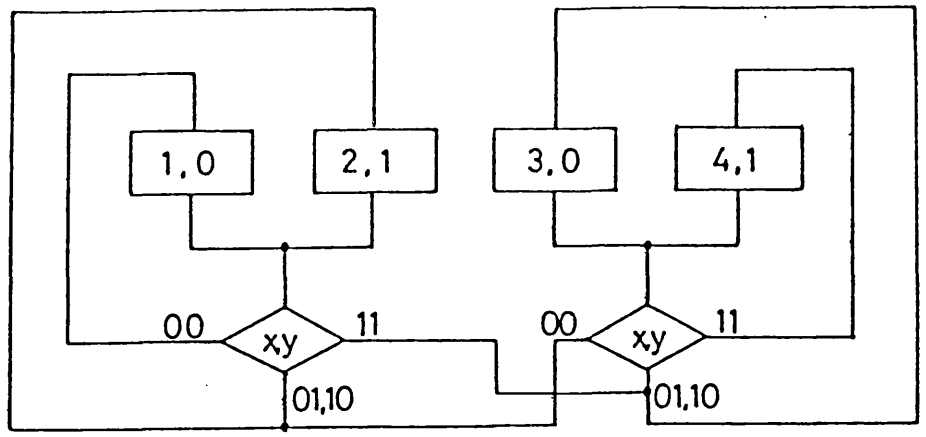
The serial full adder is an example of a system which requires latency, since when arithmetic addition is performed the result can be a larger bit word than the inputs; in the case of two binary numbers the sum requires one extra bit. The simplest way of introducing this latency into the system is to place a logic 0 value between the different words on the same input line. This then has the effect of saying if the two binary numbers are n bits long, increase them to n + 1 bit numbers where the first bit is always a logic 0. Figure 4.32(a) then shows a flow diagram which can realise the full adder system. Applying the fully specified modular realisation method as described in section 4.3.1, taking into account the additional requirements described in section 4.4.3.3 since there is more than one input, the graph and modular realisation of Figure 4.32(b) and (c) are obtained.

Note that, as the extra bit between words has been introduced, feedback is also allowed over a distance of one module without interference between input words on the same input line.

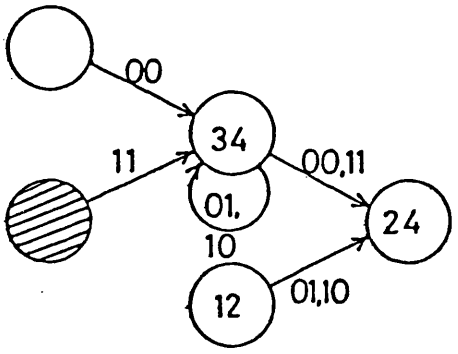
4.5 Conclusions and Further Work

It has been shown in this chapter that if a system is desired where serial word formatted logic data is to be processed then a general minimised solution can be obtained from an initial flow diagram. This solution consists of states which are the implied maximal compatibles

a)



b)



c)

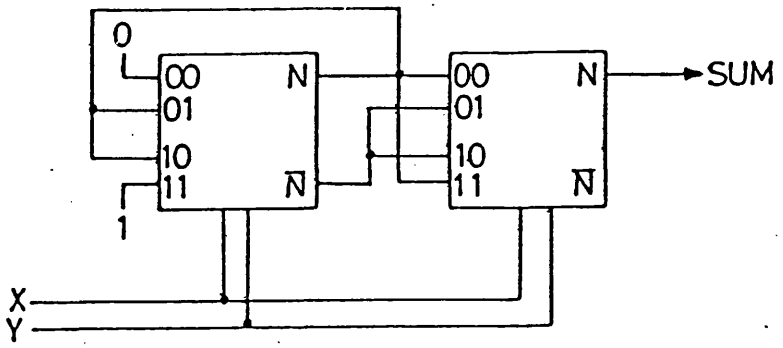


Figure 4.32 (a) Flow diagram of system which realises a full adder

(b) Graphical approach

(c) Modular realisation

of the original machine and can be used to find any hardware solution that is desirable, including the modular solution.

However, it has also been shown that if a modular solution is the desirable one, due to a number of features such as speed and testability, then there are a number of more direct methods that can be used to obtain it. If a solution is to be obtained by hand then the best method is that described in section 4.4.2.3, where the initial node of the reverse response tree is labelled with the Boolean expression of the desired function. Labels for descendant nodes are obtained by decomposing this expression about the variable x_n and then incrementing the subscripts of the variables, the process being repeated for all further descendant nodes. Nodes can then be pruned if they are labelled with a 0, a 1, or if their labels are the same or the inverse of any other nodes in the tree. This is a very simple method which is easily applied even if n is large, and it also guarantees that the minimum number of modules are required.

If a computer is to be used to obtain this same modular solution then it is more convenient if the nodes are labelled with logical values rather than Boolean expressions so that it is easier to identify equivalent or inverse labels. The best approach to this solution is to label the initial node with the truth table vector of the function transposed, i.e. converted to a row matrix from a column. Descendant nodes are then labelled with products of this matrix and the next state matrices summarised in Table 4.8. This method is also useful for incompletely specified functions as described in section 4.4.3.1.

Some of the more commonly met systems are described in section 4.4.3 such as the code converters and full adder. In a number of these systems the above approaches do not apply because, as described in

section 4.4.3.4, the output word contains more than one function; in fact, in the examples given, the output word is fully specified. The suggested approach is then that of section 4.3.1.

It was said at the beginning of the chapter that, to date, the only other approach to serial input logic has been mode-controlled logic, which was discussed in section 4.3.2. It is therefore interesting to make some comparisons. In terms of speed, the module used in mode-controlled logic has the same propagation delay as the module considered in this chapter since, by comparing Figures 4.9 and 4.12, it can be seen that the former is a special case of the latter with the M input set to a constant 0. However, the system delay, Δ clock periods, in the modular solution presented in this chapter is always minimal, i.e. $\Delta = 1$, whereas in mode-controlled logic it is at best one, and usually much more than one.

In terms of the number of modules required, the mode-controlled logic solution has some advantages since it allows feedback which is useful when highly repetitive functions are being realised. As an example, if an n variable AND function is required, then only one module is needed in mode-controlled logic, whereas n modules are required using the method developed herein. However, firstly, when feedback is used testability is reduced, and secondly, the upper limit on the number of modules requires is not specified for mode-controlled logic, whereas it is well defined by equation 4.31 for this method. It is also not clear [2] in mode-controlled logic if all functions can be realised for $n > 5$, as it can for any n by this method.

It would seem, therefore, that the modular realisation and the synthesis methods described in this chapter are viable alternatives to the existing ones and in some cases are an improvement. The area for

further work lies in its application to more conventional systems. For example, its ability to perform logical and arithmetic functions serially and at high speeds would indicate a usefulness in the arithmetic and logic unit (A.L.U.) of a computer or microprocessor. Other areas such as digital signal processing could also benefit by its application, since data in such systems is mostly in serial form.

References

1. Daws, D.C. and Jones, E.V.: "Mode-controlled serial logic systems", International Symposium on Circuits and Systems, (Chicago), pp.902-905, 27th-29th April 1981.

2. Daws, D.C.: "Serial processing of binary data using mode-controlled logic systems", Ph.D. Thesis, Essex University, 1982.

3. Jones, E.V.: "High speed transistor bistable circuits", Electron. Lett., 12, pp.375-376, 1976.

4. Daws, D.C. and Jones, E.V.: "Hardware efficient bit sequential adders and multipliers using mode-controlled logic", Electron. Lett., 16, 11, pp.434-436, May 1980.

5. Paull, M.C. and Unger, S.H.: "Minimising the number of states in incompletely specified sequential switching functions", Trans. IRE, C-8, 9, pp.356-367, Sept. 1969.

6. Rao, C.V.S. and Biswas, N.N.: "Minimisation of incompletely specified sequential machines", Trans. IEEE, C-20, 11, pp.450-452, Nov. 1975.

7. Friedman, A.D. and Menon, P.R.: "Theory and Design of Switching Circuits", (Computer Science Press - Digital Design Series, 1975).

8. Hill, F.J. and Peterson, G.R.: "Introduction of Switching Theory and Logical Design", (John Wiley and Son, 1968).

9. Lewin, D.: "Logical Design of Switching Circuits", (Thomas Nelson and Son, 1974).

10. Marcus, M.P.: "Derivation of maximal compatibles using Boolean algebra", IBM J. Res. Develop., 8, pp.537-538, Nov. 1964.

11. McCluskey, E.J.: "Minimum state sequential circuits for a restricted class of incompletely specified flow tables", Bell Syst. Tech. J., pp.1759-1768, Nov. 1962.

12. Dunworth, A. and Hartog, H.V.: "An efficient state minimisation algorithm for some special classes of incompletely specified sequential machines", Trans.IEEE, C-28, 7, pp.531-535, July 1979.

13. Eris, E.: "Minterm interchange application to digital circuit design", Ph.D. Thesis, University of Bath, 1979.

14. Hadlock, F.D. and Coates, C.L.: "Realization of sequential machines with threshold elements", Trans. IEEE, C-18, pp.428-439, May 1969.

15. Lloyd, A.M.: "Design of multiplexer universal logic module networks using spectral techniques", Proc. IEE, Pt.E, 127, 1, pp.31-36, Jan. 1980.

16. Picton, P.D.: "Clock-steering synthesis using spectral techniques", Electron. Lett., 16, 11, pp.409-411, May 1980.

17. Muzio, J. and Hurst, S.L.: "The computation of complete and reduced sets of orthogonal spectral coefficients for logic design and pattern recognition pusposes", Comput. and Electron. Eng., 5, pp.231-249, 1978.

18. Lloyd, A.M.: "Orthogonal transforms in digital logic design", Ph.D. Thesis, Bath University, 1980.

19. Newborn, M.M.: "A synthesis technique for bianry-input-binary-output synchronous sequential Moore machines", Trans. IEEE, C-17, pp.697-699, July 1968.

20. Arnold, T.F., Tan, C.-J. and New born, M.M.: "Iteratively realized sequential circuits", Trans. IEEE, C-19, 1, pp.54-66, Jan. 1970.

21. Friedman, A.D.: "Feedback in synchronous sequential switching circuits", Trans. IEEE, EG-15, 6, pp.354-367, June 1966.

22. Chao, C. and Loomis Jr., H.H.: "High rate realization of finite state machines", Trans. IEEE, C-24, 7, pp.729-741, July 1975.

23. Saluja, K.K.: "Synchronous sequential machines: a modular and testable design", Trans. IEEE, C-29, 11, pp.1020-1025, Nov. 1980.

24. Gray, F.G., Shih, L.C.C. and Thompson, R.A.: "Diagnosis of faults in modular trees", Trans. IEEE, C-28, 5, pp.342-353, May 1979.

25. Williams, G.H.: "Uniform decomposition of incompletely specified sequential machines", Trans. IEEE, C-24, 8, pp.840-843, Aug. 1975.

26. Picton, P.D.: "High speed processing of serial input logic data", Electron. Lett., 18, 3, pp.148-149, Feb. 1982.

CHAPTER 5

GENERAL CONCLUSIONS

5. General Conclusions

In Chapter 2 a method was obtained which can be used to convert a function f'(x), which can be initially expressed by a weight-threshold vector, into either of the following two functions:

i) the exclusive-or of the initial function f'(x) with one or more of the input variables,

ii) the initial function f'(x) with one of its input variables replaced by the exclusive-or of this input variable and one or more of the others.

This was done by altering the value of the weights and thresholds, and (usually) increasing the number of thresholds. Equations 2.45 and 2.65 provide the mathematical support for these operations, and it was found that it was readily applied to the weight values, but cumbersome to apply to the thresholds. However, an alternative method of determining the thresholds was presented which involved stepping through the excitation values of $\sum_{i=1}^{n} a_i x_i$ in the truth table of the final solution and comparing with the required function. The values of $\sum_{i=1}^{n} a_i x_i$ at which the function changes from a 0 to a 1, or vice versa, become the required thresholds. Thus if a function is to be converted by the repeated application of one or both of the two operations, then the equations need only be applied to the weights; the thresholds can then be determined as the final stage of the procedure.

These operations can be applied in a number of situations, but the chapter concerns itself with obtaining a multi-threshold solution of any given function. The starting point for this was the use of the Rademacher-Walsh spectrum in seeking a solution with a topology consisting of a single threshold gate with pre- and post-kernal

exclusive-or gates. The above equations then enable a conversion from this topology to that of a multi-threshold network.

In addition to this the functions that could not be converted to the single threshold topology as above, could be converrted to a multi-threshold solution if initially they can be described bby a multi-threshold gate with pre- and post-kernal exclusive-or gates. This requires that, in the Rademacher-Walsh classification scheme, the non-threshold classes have a representative function which is a multi-threshold function. Such a representative function was given for the single non-threshold class for n = 4, and it was proposed that the 27 representative functions for n = 5 also be found and listed. For n > 5, however, it is impractical to list the represemtative functions, so that functions in these classes cannot be converted to a multi-threshold solution due to a lack of representative data.

Those that can be are therefore:

i) all functions of $n \leq 8$ which can be realised by a single threshold gate after spectral translation has been applied. The limit of $n \leq 8$ is because tables of Chow's parameters are only available for up to this number of input variables.

ii) all functions of $n \leq 4$, since a representative function has been provided for the non-threshold class.

iii) the potential of all functions of $n \leq 5$, if the 27 representative functions are found and listed.

In Chapter 3 a multi-threshold logic gate was presented which could be incorporated into a C.C.D. digital processing system. Compared to existing C.C.D. logic it is similar in size, speed and complexity,

but is more powerful in that it can realise far more functions than an equivalent sized alternative.

As a separate threshold logic gate, however, i.e. a single chip with a multi-threshold logic gate only on it, it fared very poorly when compared to other proposed threshold logic gates, particularly with regard to its speed of operation. This is due to the fact that C.C.D.'s are relatively slow devices and that the major contributing factor to this speed is the charge overflow operation. The threshold logic gate relies heavily on this operation, and as the number of thresholds increases so the amount of charge overflow has to be increased, and thus the slower the device becomes. There must be a trade off therefore between the number of thresholds and hence the number of functions that can be realised by the device and the speed of operation.

However, as already stated, charge overflow is an inherent property of all C.C.D. devices, and therefore applies to all forms of C.C.D. logic, so that the threshold logic gate is still a more powerful device than the alternatives when used within a fully integrated C.C.D. system.

Finally, Chapter 4 deals with serial input logic. To date, the only other research that has been reported in this area is mode-controlled logic, and therefore this chapter was intended to broaden this field by considering serial input logic in a much more general sense.

It was shown that in the design of any serial input logic system, if the conventional approach of state tables, state assignment, etc. is used, then the number of states can be reduced by considering only

the implied maximal compatibles. This result reduces the amount of work that is involved and is also important since it defines a new class of sequential system.

Also considered in Chapter 4 is a particular modular realisation and alternative ways of obtaining it. The simplest method is shown to be the use of a reverse response tree, where the nodes are labelled with Boolean expressions. The initial node has the expression of the desired function, descendant nodes are then labelled with the decomposition about the variable x_n of this expression, and then the subscripts of all the input variables in their labels are incremented. This process is repeated until all nodes are terminal, a condition that arises when a node is labelled with a logical constant 0 or 1, or the exact equivalent or exact inverse of an already existing label on another node. The modular network is then obtained by replacing all non-terminal nodes with modules which are then connected up in the same manner as the inter-nodal connections.

An alternative labelling scheme which is useful in certain circumstances such as when a computer is used or if there are don't care terms involved is to replace the Boolean expressions with equivalent truth table vectors. Labels for descendant nodes are then obtained by taking the product of these vectors and the next state matrices.

These procedures can be extended to incorporate features such as multi-outputs or multi-inputs. The resulting networks have some desirable properties such as high speed, no inherent latency, i.e. no more than one clock period elapsing between the input of the last bit of the input word and the resulting output function, and no feedback connections so that the system is easily tested.

When compared to the mode-controlled logic networks it was shown that the speeds were the same, but that mode-controlled logic often exhibited latency and feedback was used. However, in some instances the use of feedback results in fewer modules being required such as in highly repetitive functions. However, the upper limit to the number of modules is not given, and also there is no guarantee that all functions of $n > 5$ can be realised, whereas in the network considered in Chapter 4 these two features are included.

Overall, therefore, the material presented in this thesis provides methods for extending the use of logic design. It revives the ideas of multi-threshold logic and extends them, and then finds a specific area where it may become useful. It also introduces the recently evolved topic of serial input logic, discusses it in general which results in an interesting theorem on state reduction, and suggests a modular solution which has many desirable features. It is hoped that this work will be found significant and provide a guide for further work in this area.

ACKNOWLEDGEMENTS

Acknowledgements

The author is greatly indebted to his supervisor Dr. S.L. Hurst, University of Bath, for his guidance and encouragement throughout this research.

Financial support, provided by the Science and Engineering Research Council, is gratefully acknowledged.

Finally, I would like to thank the typist, Mrs. A. Balchin, for her contribution, and to my family and friends for their unfailing support during the period of this research.

APPENDIX A

CHOW'S PARAMETERS FOR THRESHOLD FUNCTIONS OF $n \leq 6$

Chow's Parameters for Threshold Functions of n ≤ 6

Number	$\lvert b_i \rvert$	$\lvert a_i \rvert$
$n \leq 3$		
1	8 0 0 0	1 0 0 0
2	6 2 2 2	2 1 1 1
3	4 4 4 0	1 1 1 0
$n \leq 4$		
1	16 0 0 0 0	1 0 0 0 0
2	14 2 2 2 2	3 1 1 1 1
3	12 4 4 4 0	2 1 1 1 0
4	10 6 6 2 2	3 2 2 1 1
5	8 8 8 0 0	1 1 1 0 0
6	8 8 4 4 4	2 2 1 1 1
7	6 6 6 6 6	1 1 1 1 1
$n \leq 5$		
1	32 0 0 0 0 0	1 0 0 0 0 0
2	30 2 2 2 2 2	4 1 1 1 1 1
3	28 4 4 4 4 0	3 1 1 1 1 0
4	26 6 6 6 2 2	5 2 2 2 1 1
5	24 8 8 4 4 4	4 2 2 1 1 1
6	24 8 8 8 0 0	2 1 1 1 0 0
7	22 10 10 6 2 2	5 3 3 2 1 1
8	22 10 6 6 6 6	3 2 1 1 1 1
9	20 12 12 4 4 0	3 2 2 1 1 0
10	20 12 8 8 4 4	4 3 2 2 1 1
11	20 8 8 8 8 8	2 1 1 1 1 1
12	18 14 14 2 2 2	4 3 3 1 1 1
13	18 14 10 6 6 2	5 4 3 2 2 1
14	18 10 10 10 6 6	3 2 2 2 1 1
15	16 16 16 0 0 0	1 1 1 0 0 0
16	16 16 12 4 4 4	3 3 2 1 1 1
17	16 16 8 8 8 0	2 2 1 1 1 0
18	16 12 12 8 8 4	4 3 3 2 2 1
19	14 14 14 6 6 6	2 2 2 1 1 1
20	14 14 10 10 10 2	3 3 2 2 2 1
21	12 12 12 12 12 0	1 1 1 1 1 0

(cont)

Number	$\lvert b_i \rvert$							$\lvert a_i \rvert$						
$n \leq 6$														
1	64	0	0	0	0	0	0	1	0	0	0	0	0	0
2	62	2	2	2	2	2	2	5	1	1	1	1	1	1
3	60	4	4	4	4	4	0	4	1	1	1	1	1	0
4	58	6	6	6	6	2	2	7	2	2	2	2	1	1
5	56	8	8	8	8	0	0	3	1	1	1	1	0	0
6	56	8	8	8	4	4	4	6	2	2	2	1	1	1
7	54	10	10	10	6	2	2	8	3	3	3	2	1	1
8	54	10	10	6	6	6	6	5	2	2	1	1	1	1
9	52	12	12	12	4	4	0	5	2	2	2	1	1	0
10	52	12	12	8	8	4	4	7	3	3	2	2	1	1
11	52	12	8	8	8	8	8	4	2	1	1	1	1	1
12	50	14	14	14	2	2	2	7	3	3	3	1	1	1
13	50	14	14	10	6	6	2	9	4	4	3	2	2	1
14	50	14	10	10	10	6	6	6	3	2	2	2	1	1
15	50	10	10	10	10	10	10	3	1	1	1	1	1	1
16	48	16	16	16	0	0	0	2	1	1	1	0	0	0
17	48	16	16	12	4	4	4	6	3	3	2	1	1	1
18	48	16	16	8	8	8	0	4	2	2	1	1	1	0
19	48	16	12	12	8	8	4	8	4	3	3	2	2	1
20	48	12	12	12	12	8	8	5	2	2	2	2	1	1
21	46	18	18	14	2	2	2	7	4	4	3	1	1	1
22	46	18	18	10	6	6	2	9	5	5	3	2	2	1
23	46	18	14	14	6	6	6	5	3	2	2	1	1	1
24	46	18	14	10	10	10	2	7	4	3	2	2	2	1
25	46	14	14	14	10	10	6	7	3	3	3	2	2	1
26	44	20	20	12	4	4	0	5	3	3	2	1	1	0
27	44	20	20	8	8	4	4	7	4	4	2	2	1	1
28	44	20	16	16	4	4	4	6	4	3	3	1	1	1
29	44	20	16	12	8	8	4	8	5	4	3	2	2	1
30	44	20	12	12	12	12	0	3	2	1	1	1	1	0
31	44	16	16	16	8	8	8	4	2	2	2	1	1	1
32	44	16	16	12	12	12	4	6	3	3	2	2	2	1
33	42	22	22	10	6	2	2	8	5	5	3	2	1	1
34	42	22	22	6	6	6	6	5	3	3	1	1	1	1
35	42	22	18	14	6	6	2	9	6	5	4	2	2	1
36	42	22	18	10	10	6	6	6	4	3	2	2	1	1
37	42	22	14	14	10	10	2	7	5	3	3	2	2	1
38	42	18	18	18	6	6	6	5	3	3	3	1	1	1
39	42	18	18	14	10	10	6	7	4	4	3	2	2	1
40	42	18	14	14	14	14	2	5	3	2	2	2	2	1
41	40	24	24	8	8	0	0	3	2	2	1	1	0	0

(cont)

Number	$\lvert b_i \rvert$							$\lvert a_i \rvert$						
$n \leq 6$														
42	40	24	24	8	4	4	4	6	4	4	2	1	1	1
43	40	24	20	12	8	4	4	7	5	4	3	2	1	1
44	40	24	20	8	8	8	8	4	3	2	1	1	1	1
45	40	24	16	16	8	8	0	4	3	2	2	1	1	0
46	40	24	16	12	12	8	4	8	6	4	3	3	2	1
47	40	20	20	16	8	8	4	8	5	5	4	2	2	1
48	40	20	20	12	12	8	8	5	3	3	2	2	1	1
49	40	20	16	16	12	12	4	6	4	3	3	2	2	1
50	40	16	16	16	16	16	0	2	1	1	1	1	1	0
51	38	26	26	6	6	2	2	7	5	5	2	2	1	1
52	38	26	22	10	10	2	2	8	6	5	3	3	1	1
53	38	26	22	10	6	6	6	5	4	3	2	1	1	1
54	38	26	18	14	10	6	2	9	7	5	4	3	2	1
55	38	26	18	10	10	10	6	6	5	3	2	2	2	1
56	38	26	14	14	14	6	6	5	4	2	2	2	1	1
57	38	22	22	14	10	6	6	6	4	4	3	2	1	1
58	38	22	22	10	10	10	10	3	2	2	1	1	1	1
59	38	22	18	18	10	10	2	7	5	4	4	2	2	1
60	38	22	18	14	14	10	6	7	5	4	3	3	2	1
61	38	18	18	18	14	14	2	5	3	3	3	2	2	1
62	36	28	28	4	4	4	0	4	3	3	1	1	1	0
63	36	28	24	8	8	4	4	6	5	4	2	2	1	1
64	36	28	20	12	12	4	0	5	4	3	2	2	1	0
65	36	28	20	12	8	8	4	7	6	4	3	2	2	1
66	36	28	16	16	12	4	4	6	5	3	3	2	1	1
67	36	28	16	12	12	8	8	8	7	4	3	3	2	2
68	36	24	24	12	12	4	4	7	5	5	3	3	1	1
69	36	24	24	12	8	8	8	4	3	3	2	1	1	1
70	36	24	20	16	12	8	4	8	6	5	4	3	2	1
71	36	24	20	12	12	12	8	5	4	3	2	2	2	1
72	36	24	16	16	16	8	8	4	3	2	2	2	1	1
73	36	20	20	20	12	12	0	3	2	2	2	1	1	0
74	36	20	20	16	16	12	4	6	4	4	3	3	2	1
75	34	30	30	2	2	2	2	5	4	4	1	1	1	1
76	34	30	26	6	6	6	2	7	6	5	2	2	2	1
77	34	30	22	10	10	6	2	8	7	5	3	3	2	1
78	34	30	18	14	14	2	2	7	6	4	3	3	1	1
79	34	30	18	14	10	6	6	9	8	5	4	3	2	2
80	34	30	14	14	10	10	10	7	6	3	3	2	2	2
81	34	26	26	10	10	6	6	5	4	4	2	2	1	1
82	34	26	22	14	14	6	2	9	7	6	4	4	2	1

(cont)

Number	$\lvert b_i \rvert$							$\lvert a_i \rvert$						
$n \leq 6$														
83	34	26	22	14	10	10	6	6	5	4	3	2	2	1
84	34	26	18	18	14	6	6	5	4	3	3	2	1	1
85	34	26	18	14	14	10	10	6	5	4	3	3	2	2
86	34	22	22	18	14	10	2	7	5	5	4	3	2	1
87	34	22	22	14	14	14	6	4	3	3	2	2	2	1
88	34	22	18	18	18	10	6	5	4	3	3	3	2	1
89	32	32	32	0	0	0	0	1	1	1	0	0	0	0
90	32	32	28	4	4	4	4	4	4	3	1	1	1	1
91	32	32	24	8	8	8	0	3	3	2	1	1	1	0
92	32	32	20	12	12	4	4	5	5	3	2	2	1	1
93	32	32	16	16	16	0	0	2	2	1	1	1	0	0
94	32	32	16	16	8	8	8	4	4	2	2	1	1	1
95	32	32	12	12	12	12	12	3	3	1	1	1	1	1
96	32	28	28	8	8	8	4	6	5	5	2	2	2	1
97	32	28	24	12	12	8	4	7	6	5	3	3	2	1
98	32	28	20	16	16	4	4	6	5	4	3	3	1	1
99	32	28	20	16	12	8	8	7	6	5	4	3	2	2
100	32	28	16	16	12	12	12	5	4	3	3	2	2	2
101	32	24	24	16	16	8	0	4	3	3	2	2	1	0
102	32	24	24	16	12	12	4	5	4	4	3	2	2	1
103	32	24	20	20	16	8	4	6	5	4	4	3	2	1
104	32	24	20	16	16	12	8	7	6	5	4	4	3	2
105	32	20	20	20	20	8	8	3	2	2	2	2	1	1
106	30	30	30	6	6	6	6	3	3	3	1	1	1	1
107	30	30	26	10	10	10	2	5	5	4	2	2	2	1
108	30	30	22	14	14	6	6	4	4	3	2	2	1	1
109	30	30	18	18	18	2	2	5	5	3	3	3	1	1
110	30	30	18	18	10	10	10	3	3	2	2	1	1	1
111	30	30	14	14	14	14	14	2	2	1	1	1	1	1
112	30	26	26	14	14	10	2	6	5	5	3	3	2	1
113	30	26	22	18	18	6	2	7	6	5	4	4	2	1
114	30	26	22	18	14	10	6	8	7	6	5	4	3	2
115	30	26	18	18	14	14	10	6	5	4	4	3	3	2
116	30	22	22	22	18	6	6	4	3	3	3	2	1	1
117	30	22	22	18	18	10	10	5	4	4	3	3	2	2
118	28	28	28	12	12	12	0	2	2	2	1	1	1	0
119	28	28	24	16	16	8	4	5	5	4	3	3	2	1
120	28	28	20	20	20	4	0	3	3	2	2	2	1	0
121	28	28	20	20	12	12	8	4	4	3	3	2	2	1
122	28	28	16	16	16	16	12	3	3	2	2	2	2	1
123	28	24	24	20	20	4	4	5	4	4	3	3	1	1

(cont)

Number	$\lvert b_i \rvert$							$\lvert a_i \rvert$						
$n \leq 6$														
124	28	24	24	20	16	8	8	6	5	5	4	3	2	2
125	28	24	20	20	16	12	12	7	6	5	5	4	3	3
126	26	26	26	18	18	6	6	3	3	3	2	2	1	1
127	26	26	22	22	22	2	2	4	4	3	3	3	1	1
128	26	26	22	22	14	10	10	5	5	4	4	3	2	2
129	26	26	18	18	18	14	14	4	4	3	3	3	2	2
130	26	22	22	22	14	14	14	4	3	3	3	2	2	2
131	24	24	24	24	24	0	0	1	1	1	1	1	0	0
132	24	24	24	24	12	12	12	2	2	2	2	1	1	1
133	24	24	20	20	16	16	16	5	5	4	4	3	3	3
134	22	22	22	18	18	18	18	3	3	3	2	2	2	2
135	20	20	20	20	20	20	20	1	1	1	1	1	1	1

For tables of Chow's parameters of $n \leq 7$, and $n \leq 8$ see Winder [7] and Muroga, Tsuboi and Baugh [8], referenced in Chapter 2.

APPENDIX B

RADEMACHER-WALSH SPECTRAL CLASSIFICATION OF FUNCTIONS OF $n \leq 4$

Canonic function	Spectral Coefficients																Linear-Separable (threshold) or not
	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}	
1	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Yes
2	14	2	2	2	2	-2	-2	-2	-2	-2	-2	2	2	2	2	-2	Yes
3	12	4	4	4	0	-4	-4	0	-4	0	0	4	0	0	0	0	Yes
4	10	6	6	2	2	-6	-2	-2	-2	-2	2	2	2	-2	-2	2	Yes
5	8	8	8	0	0	-8	0	0	0	0	0	0	0	0	0	0	Yes
6	8	8	4	4	4	-4	-4	-4	0	0	0	0	0	0	-4	4	Yes
7	6	6	6	6	6	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	6	Yes
8	4	4	4	4	4	4	4	-4	-4	4	-4	-4	-4	4	4	-4	No

The canonic spectral classification of all binary functions of $n \leq 4$ under the full Rademacher-Walsh invariance operations.

Note: Entries 1, 3 and 5 are for functions of $n < 4$ variables, whilst entries 2, 4, 6, 7 and 8 are for functions of exactly $n = 4$ variables. Also entries 1 to 7 are all linearly-separable functions, with entry 8 as the only non-linearly-separable entry.

APPENDIX C

COPIES OF PUBLISHED MATERIAL

CLOCK-STEERING SYNTHESIS USING SPECTRAL TECHNIQUES

Indexing terms: Logic, Switches

A technique is described which uses the Rademacher-Walsh spectrum to obtain the spectra, and consequently the Boolean form, of the clock-steering functions of bistables. Its usefulness over standard methods is derived from the fact that its simplicity is independent of the number of input variables. Being numerically based, it also is particularly relevant for c.a.d. adoption.

Introduction: Type *D*, *JK*, *RS* and *T* clocked bistables are well-known items of sequential logic design. Where there are only a few input variables, derivation of the appropriate clock-steering input functions to the bistables from the next-state equations is a straightforward procedure. Previous authors[1-4] generally use one of two methods, involving either the use of Karnaugh maps or the characteristic equations of the bistables. These characteristic equations are as follows:

Type *D*: Next-state output $x_i' = D$ (1)

Type *JK*: Next-state output $x_i' = \{x_i J + x_i \bar{K}\}$ (2)

Type *RS*: Next-state output $x_i' = \{\bar{R}(S + x_i)\}$ (3)
where *R* and *S* are disjoint;

Type *T*: Next-state output $x_i' = \{\bar{x}_i T + x_i \bar{T}\} = \{x_i \oplus T\}$ (4)

Note that x_i represents the present state of the true output of the bistable circuit, and x_i' the resultant next-state of the same output point one clock pulse later. Fig. 1 shows a typical assembly to which the above equations may be applied.

As the number of input variables increases, however, it becomes increasingly difficult to obtain the input functions using the aforementioned methods, as both Karnaugh mapping and algebraic manipulation become prohibitively tedious. It is proposed, therefore, that spectral techniques be applied to overcome this difficulty.

The Rademacher-Walsh spectrum $S]$ of an n-variable combinational logic function is an alternative representation of its truth-table vector $F]$ of 2^n elements. The spectrum is formed by the multiplication of $F]$ with a $2^n \times 2^n$ transform matrix $[T]$; that is

$$[T]F] = S]$$

where the square transform matrix $[T]$ is (usually) the Rademacher-Walsh transform, and $F]$ is the truth table of the given function, but recoded $\langle +1, -1 \rangle$ from the more conventional $\langle 0, 1 \rangle$. The resulting numbers in $F]$ uniquely define the given function, and represent a set of correlation coefficients between the individual inputs and combination of inputs of the function, and the function output. The theory and use of spectral methods in the design of combinational logic circuits may be found in published literature.[5-8] The inverse transformation from the spectral domain back to the two-valued domain, namely

$$[T]^{-1}S] = F]$$

is readily available.

Method: Let the next-state equation x_i' have the following Rademacher-Walsh spectrum:[5-7]

$$r_0, r_1, r_2, r_3, \ldots, r_i, \ldots, r_n, r_{12}, r_{13}, \ldots, r_{n-1}r_n, \ldots, r_{123\ldots n}$$

Now consider each type of bistable separately.

(i) *Type D:* This is a trivial case which does not require consideration, since by definition of the type *D*, x_i' is always equal to *D*.

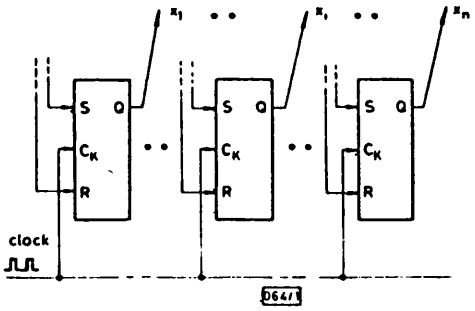


Fig. 1 *Typical accumulator or register assembly: type RS storage elements shown, with R and S clock-steering logic $f(x_1, \ldots, x_n)$ to be designed for each R and S input. (Similarly for type D, T and JK storage elements)*

(ii) *Type JK:* From eqn. 2 it can be seen that J and K are the reduced functions obtained from a Shannon decomposition about x_i. From previously published information concerning the spectra of decomposed functions,[7] it follows that, if J and K have the following spectra, respectively,

J: $J_0, J_1, J_2, \ldots, J_n, J_{12}, J_{13}, \ldots, J_{n-1}, J_n, \ldots, J_{12\ldots n}$

K: $K_0, K_1, K_2, \ldots, K_n, K_{12}, K_{13}, \ldots, K_{n-1}, K_n, \ldots, K_{12\ldots n}$

where no subscript contains i, then these spectra are obtainable from the full spectrum of x_i' as follows:

$$\left.\begin{aligned} J_j &\equiv \tfrac{1}{2}(r_{ij} + r_j) \\ K_j &\equiv \tfrac{1}{2}(r_{ij} - r_j) \end{aligned}\right| \qquad (5)$$

where j = all possible subscripts not involving i, including $j = 0$. Application of the inverse transform or by using prime implicant extraction techniques[9,10] will then yield the Boolean solutions for J and K (see the example later).

(iii) *Type RS:* Eqn. 3 can be manipulated to give

$$x_i' = \{\bar{x}_i S + x_i R\} \qquad (6)$$

since S and R are disjoint. From this it can be seen, by comparison with eqn. 2, that if J and K are disjoint then they are equivalent to S and R, respectively. It is therefore necessary to test whether the JK solution would be disjoint before accepting such a solution for S and R. This can be done as follows:

If J and K are disjoint, we have

$$\frac{1}{2^{n-1}}\{[J_0, J_1, \ldots, J_{12\ldots n}]K_0\} = \{-2^{n-1} + J_0 + K_0\}$$
$$K_1$$
$$\vdots$$
$$K_{12\ldots n}$$

Substituting for all J_j, K_j from eqn. 5, we obtain

$$\sum_j \{r_{ij}^2 - r_j^2\} = \{-2^{2n} + 2^{n+1} r_i\} \qquad (7)$$

Therefore if this equation is correct then S_j and R_j are equivalent to J_j and K_j in eqn. 5. If not, then use the modified expressions given in eqn. 8, which ensures that S and R are disjoint:

$$\left.\begin{aligned} S &= J\bar{x}_i \\ R &= Kx_i \end{aligned}\right| \qquad (8)$$

This yields the following:

$$\left.\begin{aligned} S_0 &= \tfrac{1}{2}(r_i + r_0) + 2^{n-1} & R_0 &= \tfrac{1}{2}(r_i - r_0) + 2^{n-1} \\ S_j &= \tfrac{1}{2}(r_{ij} + r_j) & R_j &= \tfrac{1}{2}(r_{ij} - r_j) \\ S_i &= \tfrac{1}{2}(r_i + r_0) - 2^{n-1} & R_i &= \tfrac{1}{2}(r_0 - r_i) + 2^{n-1} \\ &= S_0 - 2^n & &= -R_0 + 2^n \\ S_{ij} &= \tfrac{1}{2}(r_{ij} + r_i) = S_j & R_{ij} &= \tfrac{1}{2}(r_j - r_{ij}) = -R_j \end{aligned}\right| \qquad (9)$$

where j = all possible subscripts not involving i, including $j = 0$.

(iv) *Type T:* This solution can be obtained directly from eqn. 4. Since the exclusive-OR relationship is commutative, then

$$T = \{x_i \oplus x_i'\} \qquad (10)$$

In the spectral domain, this is equivalent to a disjoint spectral translation,[5,6] that is, in all 2^n spectral coefficients, append i if it does not appear in the subscript and delete it if it is already present.

Restated, this is

$$\left.\begin{aligned} &r_i \leftrightarrow r_0 \\ &r_{ij} \leftrightarrow r_j \\ &r_{ijk} \leftrightarrow r_{jk} \quad \text{etc.} \end{aligned}\right| \qquad (11)$$

Example: The next-state equation for circuit x_4 is

$$x_4' = \{x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_3\bar{x}_4 + x_1x_2x_3x_4 + \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3x_4\}$$

The Rademacher-Walsh spectrum is

r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
0	0	0	0	0	0	8	0	0	0	0	8	0	8	0	−8

From eqn. 5 we have a JK solution:

	r_0	r_1	r_2	r_3	r_{12}	r_{13}	r_{23}	r_{123}
J	0	0	0	0	0	8	0	0
K	0	0	0	0	0	0	0	−8

whence

$$J = \{x_1 \oplus x_3\} = \{\bar{x}_1x_3 + x_1\bar{x}_3\}$$
$$K = \{\overline{x_1 \oplus x_2 \oplus x_3}\}$$
$$= \{\bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3\}$$

For an RS solution, first check whether J and K are disjoint using eqn. 7:

$$\{64 + 64 - 64 - 64\} = \{-256 + (32 \times 0)\} \quad ?$$
$$0 \neq -256$$

Clearly J and K are not disjoint, so, from eqn. 9,

	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
S	8	0	0	0	−8	0	8	0	0	0	0	0	0	8	0	0
R	8	0	0	0	8	0	0	0	0	0	0	−8	0	0	0	8

whence it follows that:

$$S = \{(x_1 \oplus x_3)\bar{x}_4\} = \{\bar{x}_4(\bar{x}_1x_3 + x_1\bar{x}_3)\}$$
$$R = \{(\overline{x_1 \oplus x_2 \oplus x_3})x_4\}$$
$$= \{x_4(\bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3)\}$$

For a type T solution, from eqn. 11 we have:

	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
T	0	0	0	0	0	0	8	0	0	0	0	−8	0	8	0	8

whence it follows that

$$T = \{\bar{x}_4(x_1 \oplus x_3) + x_4(\overline{x_1 \oplus x_2 \oplus x_3})\}$$
$$= \{x_1(\bar{x}_3\bar{x}_4 + x_2\bar{x}_3 + \bar{x}_2x_3x_4) + \bar{x}_1(x_2x_3 + x_3\bar{x}_4 + \bar{x}_2\bar{x}_3x_4)\}$$

Note that, for convenience, this example has been of four variables only, and hence within the capabilities of the nonspectral design methods. However, the simplicity of manipulation of the spectral data for larger functions than can be handled by

previous methods should be apparent from this example. Its ease of incorporation into c.a.d. programs should also be apparent.

Acknowledgments: This work has been pursued under UK Science Research Council Studentship 79310262.

P. D. PICTON

School of Electrical Engineering
University of Bath
Claverton Down, Bath BA2 7AY, England

8th April 1980

References

1 MORRIS MANO, M.: 'Digital logic and computer design' (Prentice-Hall, 1979)
2 BENNETTS, R. G.: 'Switching theory in the design of logical systems', *Design Electronics*, 1971, 8, pp. 54–59
3 BENNETTS, R. G.: 'Algebraic models for clocked flip-flops', *Electron. Eng.*, 1978, 50, pp. 59–62
4 OBERMAN, R. M. M.: 'The *JK* gate', *IEEE Trans.*, 1976, C-25, pp. 1156–1159
5 EDWARDS, C. R.: 'The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis', *Ibid.*, 1975, C-24, pp. 48–62
6 HURST, S. L.: 'Logical processing of digital signals' (Crane-Russak, NY, and Edward Arnold, London, 1978)
7 MUZIO, J. C., and HURST, S. L.: 'The computation of complete and reduced sets of orthogonal spectral coefficients for logic design and pattern recognition purposes', *Comput. & Electron. Eng.*, 1978, 5, pp. 231–249
8 LLOYD, A. M.: 'Design of multiplexer universal-logic-module networks using spectral techniques', *IEE Proc. E, Comput. & Digital Tech.*, 1980, 127, (1), pp. 31–36
9 BESSLICH, P. W.: 'On the Walsh-Hadamard transform and prime implicant extraction', *IEEE Trans.*, 1978, ECM-20, pp. 516–519
10 LLOYD, A. M.: 'Spectral addition techniques for the synthesis of multivariable logic networks', *IEE J. Comput. & Digital Tech.*, 1978, 1, pp. 152–164

0013-5194/80/110409-03$1.50/0

Realisation of multithreshold threshold logic networks using the Rademacher-Walsh transform

P.D. Picton, B.Sc.

Indexing terms: *Logic, Multithreshold threshold elements, Rademacher-Walsh transform*

Abstract: A method is proposed by which binary logic functions are resolved into multithreshold form. It is based upon an already existing design algorithm which involves spectral translations, and can be easily adopted for CAD implementation.

List of symbols

n = number of independent binary input variables
x_i = ith input variable
a_i = ith weight of a multithreshold threshold element
t_i = ith threshold of a multithreshold threshold element
m = number of thresholds of a multithreshold threshold element
$f_i(x)$ = ith output of a multithreshold threshold element
$f(x)$ = required function or output
$f'(x)$ = "kernal" function
a_i' = ith weight of the kernal function
t_i' = ith threshold of the kernal function
p = number of thresholds of the kernal function
$f_i'(x)$ = ith output of the kernal function
$\oplus$ = Exclusive-OR or modulo-2 sum

1 Introduction

Previous research work in the area of majority and threshold logic has been largely mathematical [1–6], being constrained by the nonavailability of viable threshold-logic and other non-Boolean (nonvertex) logic gates. The potential power and discrimination of threshold-logic relationships has been extensively considered, but without viable circuit realisations the commercial dominance of the Boolean basis of present digital systems cannot be challenged.

However, recent research and development in semiconductor technology has increased the possibility of commercial circuit realisations for non-Boolean logic gates [7, 8]. Developments in both I^2L and CCD technology appear particularly relevant, CCD in particular appearing to offer a direct promise towards 'weighted' digital signals and thresholds such as are involved in the summation and detection requirements of threshold logic. Hence it is not irrelevant to again consider synthesis techniques using non-Boolean relationships as a parallel line of research to the wider areas of semiconductor technology research.

On the theoretical system synthesis side, previous work by Edwards and others [4, 9] has shown how the Rademacher-Walsh spectral coefficients may be used to classify functions, and that all Boolean functions may be synthesised using optimised threshold-logic gates together with a small number of Exclusive-OR gates. As an alternative approach, Haring and others [10–12] have shown that all functions can be realised with a multithreshold threshold-logic element, and in particular have tabulated all 221 optimised multithreshold functions corresponding to the NPN classification for $n \leq 4$.*

*The NPN classification is a means of grouping similar topology functions together which are related by the following three operations: Negation (complementation) of the input variables, Permutation of the input variables, and Negation of the overall function [1, 4].

Paper 1298E, received 3rd November 1980

Mr. Picton is with the School of Electrical Engineering, University of Bath, Claverton Down, Bath, BA2 7AY, England

Thus in total this tabulation covers all possible 65536 functions of four or fewer variables.

These two approaches are illustrated in Fig. 1. Two features in connection with the multithreshold topology of Fig. 1*b*, however, should be particularly noted, namely:

(*a*) The multithreshold outputs progressively switch from logic 0 to logic 1 with increase in gate input summation $\sum_{i=1}^{n} a_i x_i$, as shown in Fig. 2*a*. The multi-input Exclusive-OR interrogating these outputs therefore provides a final output as in Fig. 2*b*.

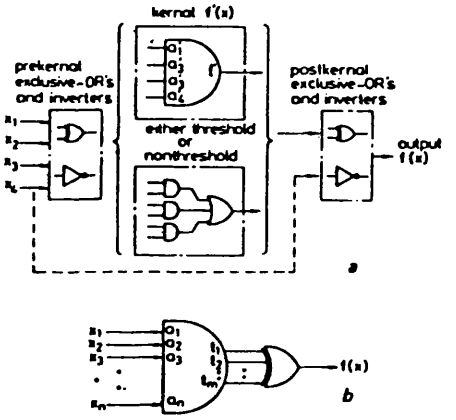


Fig. 1 *Two threshold-logic approaches to logic synthesis*

a Spectral translation design topology, generating a kernal function with pre- and post-Exclusive-OR and/or inverter gates
b Multithreshold element design topology, with post-threshold combinational requirements only

(*b*) It will be seen that the output Exclusive-OR logic never has to cater for the output condition $t_{i+1} = 1$ with $t_i = 0$, and hence this output logic may be realised by half-Exclusive-OR relationships [4], i.e. $[f_i(x)\overline{f_{i+1}(x)}]$ rather than $[f_i(x)\overline{f_{i+1}(x)} + \overline{f_i(x)}f_{i+1}(x)]$. Therefore instead of the full Exclusive-OR gates required by the spectral translation synthesis methods of Fig. 1*a*, much simpler half-Exclusive-OR gates such as shown in Fig. 2*c* are possible with multithreshold synthesis.

This paper therefore pursues this latter approach, with its simpler Exclusive relationships, but with the possibly more complex multithreshold kernal. Circuit designs for multi-threshold gates have been considered by previous authors [2–4, 13], particularly in Digital-Summation-Threshold-Logic (DSTL) array form, but the possibility of a more viable realisation in CCD technology arises if present developments in this area continue. In particular CCD has inherent properties which offer the possibility of including more than one output threshold without greatly increasing

0143-7062/81/030107+07 $01.50/0

Table 1: Canonic spectral classification of all binary functions of $n \leq 4$ under the full Rademacher-Walsh invariance operations

Canonic function	Spectral coefficients																Linear-separable (threshold) or not
	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}	
1	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Yes
2	14	2	2	2	2	−2	−2	−2	−2	−2	2	2	2	2	2	−2	Yes
3	12	4	4	4	0	−4	−4	0	−4	0	0	4	0	0	0	0	Yes
4	10	6	6	2	2	−6	−2	−2	−2	−2	2	2	2	−2	−2	2	Yes
5	8	8	8	0	0	−8	0	0	0	0	0	0	0	0	0	0	Yes
6	8	8	4	4	4	−4	−4	−4	0	0	0	0	0	0	−4	4	Yes
7	6	6	6	6	6	−2	−2	−2	−2	−2	−2	−2	−2	−2	−2	6	Yes
8	4	4	4	4	4	4	4	−4	−4	4	−4	−4	−4	4	4	−4	No

Entries 1, 3 and 5 are for functions of $n < 4$ variables, whereas entries 2, 4, 6, 7 and 8 are for functions of exactly $n = 4$ variables. Also entries 1 to 7 are all linearly-separable functions, with entry 8 as the only nonlinearly-separable entry.

the gate complexity, which is a powerful attraction over previous concepts. This is currently being researched by the author.

2 Developments

The tabulated methods of synthesis noted earlier [10] are restricted to a maximum of four input variables ($n = 4$), due to the nonavailability of tabulated data for $n > 4$. The developments disclosed herewith, however, extend the spectral translation methods of Edwards *et al.* [9] to give multithreshold realisations, giving multithreshold results comparable to the $n \leq 4$ tabulated results of Haring, but which may be extended to functions of $n > 4$.

Consider the case of four-variable combinational Boolean functions. Using Rademacher-Walsh classification procedures, all 65536 functions of $n \leq 4$ can be compressed into the eight positive canonic entries detailed in Table 1 [4, 9]. Thus all $n \leq 4$ functions can be made with the network topology shown in Fig. 1*a*, the pre- and post- kernal operations of Exclusive-OR and Inversion corresponding to the operations involved in the spectral classification procedure [9]. Only one design of nonthreshold kernal is required for $n \leq 4$, to cater for functions covered by the classification entry 8 of Table 1.

For $n > 4$ there remains the same possible threshold or nonthreshold topology of Fig. 1*a*, but now there is a requirement for more than one design of nonthreshold kernal to cover the many nonlinearly-separable functions which are present in $n > 4$ canonic classifications. However, as already noted, any Boolean function of n variables, where n can be any value, can be made with the single multi-output threshold topology of Fig. 1*b*, provided it has appropriate weights and sufficient thresholds and half-Exclusive-OR relationships on the outputs to distinguish each threshold output [10, 13]. This will be our objective in the following developments.

Conversion from the first to the second network topologies shown in Fig. 1 involves the three following operations:

(*a*) conversion into multithreshold form of functions requiring disjoint spectral translation, that is the postkernal Exclusive-OR of the kernal function output $f'(x)$ with one (or more) of the x_i input variables

(*b*) conversion into multithreshold form of functions requiring input spectral translations, that is prekernal Exclusive-OR's of one or more of the x_i input variables to create a new input variable set to the function $f'(x)$

(*c*) conversion into multithreshold form of any nonthreshold kernal function $f'(x)$.

Any given function $f(x)$ may require some or all of these three operations to be applied if it is not inherently a single or multithreshold function in its own right.

To consider the above operations for our present purposes, first let us recall that the Rademacher-Walsh spectrum $S]$ of a n-variable combinational logic function $f(x)$ is an alternative representation of its truth-table vector $F]$ of 2^n elements [4, 14]. The spectrum is formed by the multiplication of $F]$ with a $2^n \times 2^n$ transform matrix, $[T]$, that is:

$$[T]F] = S]$$

where the square transform matrix $[T]$ is (usually) the Rademacher-Walsh transform, and $F]$ is the truth table of the given function, but recoded $(+1, -1)$ from the more conventional $(0, 1)$. The resulting numbers in $S]$, the spectral coefficients, uniquely define the given function $f(x)$, and represent a set of correlation coefficients between the individual inputs and combinations of inputs of the function, and the function output. If a function is a threshold function, then the primary coefficients, i.e. the first $n + 1$ coefficients, are equivalent to the Chow parameters found in published tables [1, 4], and can be used to read off the weights and threshold required to realise the function.

The theory and use of spectral methods in the design of combinational logic circuits may be found in published literature [4, 9, 15]. The inverse transformation from the spectral domain back to the two-valued domain, namely

$$[T]^{-1}S] = F]$$

is readily available.

The relationships of the spectral coefficients of functions $f'(x)$ and $f(x)$, see Fig. 1*a*, under the above three operations will now be considered.

2.1 *Disjoint spectral translation*

Disjoint spectral translation is the modification of the output of the network $f'(x)$, where $f(x) = f'(x) \oplus x_i$; this operation results in the interchange of pairs of primary and secondary coefficient values as follows between the spectrum of $f'(x)$ and $f(x)$ as follows:

$$r_i \leftrightarrow r_0$$

$$r_{ij} \leftrightarrow r_j$$

$$r_{ijk} \leftrightarrow r_{jk} \text{ etc.}$$

This may be restated as: in all 2^n coefficients append i if it does not appear and delete it if it is already present.

In this situation we consider the function $f(x)$ is not

linearly separable but the function $f'(x)$ is. Thus $f'(x)$ can be represented by a weight-threshold vector of

$$f'(x) \equiv a'_1, a'_2, \ldots, a'_i, \ldots, a'_n; t'$$

If a multithreshold solution can be obtained it will be in the form of

$$f(x) = \{f_1(x) \bullet f_2(x) \bullet f_3(x) \ldots \bullet f_m(x)\}$$

which in weight-threshold vector form is

$$f(x) \equiv a_1, a_2, \ldots, a_i, \ldots, a_n; t_1, t_2, t_3, \ldots, t_m$$

where $t_1 < t_2 < t_3 < \ldots < t_m$, see Fig. 2
It can be shown that a possible solution is

$$f_1(x) = f'(x) + x_i$$

$$f_2(x) = f'(x)x_i$$

since

$$f_1(x) \bullet f_2(x) = (f'(x) + x_i) \bullet f'(x)x_i = f'(x) \bullet x_i$$

The weight-threshold vector corresponding to these functions is

$$f(x) \equiv a'_1, a'_2, \ldots, a'_i + c, \ldots, a'_n; t', t' + c$$

where c (assuming all weights are positive) is given by

$$\left.\begin{aligned} c &\geq t' - a'_i \\ c &\geq \sum_{L=1}^{n} a'_L + 1 - a'_i - t' \end{aligned}\right\} \text{both must be satisfied}$$

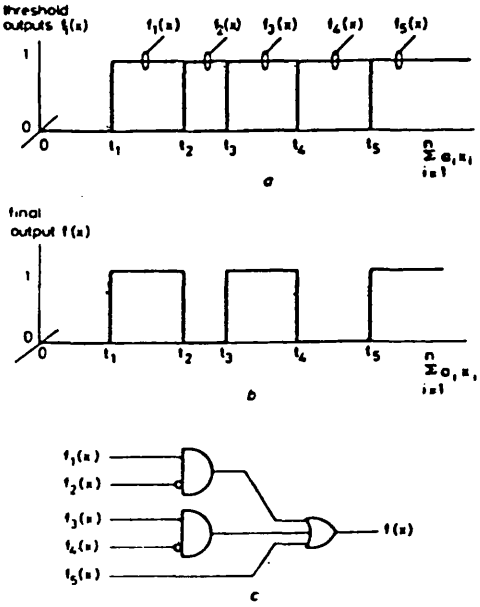


Fig. 2 *Multithreshold relationships of Fig. 1.*

a Individual outputs $f_1(x)$ at t_1, $f_2(x)$ at t_2, of the multithreshold gate

b Exclusive-OR output from the multithreshold outputs $f_1(x)$, $f_2(x)$, ...

c The use of half-Exclusive-OR gates $\{f_i(x)\overline{f_{i+1}(x)}\}$ instead of normal Exclusive-OR gates $\{f_i(x)\overline{f_{i+1}(x)} + \overline{f_i(x)}f_{i+1}(x)\}$ to realise the final output $f(x)$

In other words, the original weight a'_i is raised by this amount c and another threshold is added equal to the original threshold t' raised by the same amount c.

The same method applies when more than one input-variable is fed forward and Exclusive-OR'ed with the function $f'(x)$, e.g. with three input variables fed forward we have

$$f(x) = f'(x) \bullet x_i \bullet x_j \bullet x_k$$

Here we find that

$$f(x) = f_1(x) \bullet f_2(x) \bullet f_3(x) \bullet f_4(x)$$

where

$$f_1(x) = f'(x) + x_i + x_j + x_k$$

$$f_2(x) = f'(x)(x_i + x_j + x_k) + x_i(x_j + x_k) + x_j x_k$$

$$f_3(x) = f'(x)[x_i(x_j + x_k) + x_j x_k] + x_i x_j x_k$$

$$f_4(x) = f'(x)x_i x_j x_k$$

or that if

$$f'(x) \equiv a'_1, a'_2, \ldots, a'_i, a'_j, a'_k, \ldots, a'_n; t'$$

then

$$f(x) \equiv a'_1, a'_2, \ldots, a'_i + c, a'_j + c, a'_k + c, \ldots, a'_n$$
$$t', t' + c, t' + 2c, t' + 3c$$

where c (assuming all weights are positive) is given by

$$\left.\begin{aligned} c &\geq t' - a'_i \text{ and } c \geq \sum_{L=1}^{n} a'_L + 1 - a'_i - t' \\ c &\geq t' - a'_j \text{ and } c \geq \sum_{L=1}^{n} a'_L + 1 - a'_j - t' \\ c &\geq t' - a'_k \text{ and } c \geq \sum_{L=1}^{n} a'_L + 1 - a'_k - t' \end{aligned}\right\} \text{all must be satisfied}$$

Example 1 in the following Section shows the application of this numerical procedure to determine the required input weights and output thresholds.

2.2 *Input spectral translation*

Input spectral translation [9] is the replacement of any input variable x_i by the Exclusive-OR signal $x_i \bullet x_j, i \neq j \neq 0$; this operation results in the interchange of all pairs of primary and secondary coefficient values which contain i in their subscripts as follows:

$$r_i \leftrightarrow r_{ij}$$

$$r_{ik} \leftrightarrow r_{ijk} \quad \text{etc.}$$

with r_0 and all other coefficients which do not contain i in their subscripts remaining unchanged. This may be restated as: in all coefficients which contain the subscript i delete j if it also appears and append it if it does not.

In this situation we consider the function $f(x)$ is not linearly separable but $f'(x)$ is. Thus $f'(x)$ can be represented by the weight-threshold vector thus:

$$f'(x) \equiv a'_1, a'_2, \ldots, a'_i, a'_j, \ldots, a'_n; t'$$

The multithreshold solution would take the form of

$$f(x) = \{f_1(x) \oplus f_2(x) \oplus f_3(x) \ldots \oplus f_m(x)\}$$

A possible solution to this is

$$f_1(x) = f'_{00}(x) + x_i + x_j$$

$$f_2(x) = x_i + x_j$$

$$f_3(x) = x_i f'_{10}(x) + x_j f'_{11}(x) + x_i x_j$$

$$f_4(x) = x_i x_j$$

$$f_5(x) = x_i x_j f'_{01}(x)$$

where $f'_{00}(x)$, $f'_{01}(x)$, $f'_{10}(x)$ and $f'_{11}(x)$ are obtained from a Shannon decomposition of $f'(x)$ as follows:

$$f'(x) = \bar{x}_i \bar{x}_j f'_{00}(x) + \bar{x}_i x_j f'_{01}(x) + x_i \bar{x}_j f'_{10}(x) + x_i x_j f'_{11}(x)$$

The corresponding weight-threshold vector for this solution (assuming all weights are positive) is

$$f(x) \equiv a'_1, a'_2, \ldots, \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1, \sum_{L=1}^{n} a'_L - a'_i + 1, \ldots . a'_n; t_1, t_2, t_3, t_4, t_5$$

The thresholds t_1 to t_5 can be either calculated separately by simply stepping through the truth table of the function or, alternatively, values can be computed as follows:

$$t_1 = t' - \underbrace{(t' - \sum_{L=1}^{n} a'_L + a'_i + a'_j - 1)}$$

included if $t' > \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1$

$$t_2 = \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1$$

$$t_3 = t' + \sum_{L=1}^{n} a'_L - 2a'_i - a'_j + 1 + \underbrace{(a'_i - t')}$$

included if $a'_i > t'$

$$t_4 = 2 \sum_{L=1}^{n} a'_L - 2a'_i - a'_j + 2$$

$$t_5 = t' + 2 \sum_{L=1}^{n} a'_L - 2a'_i - 2a'_j + 2 + \underbrace{(a'_j - t')}$$

included if $a'_j > t'$

If the original function $f'(x)$ has more than one threshold to start with, then each original threshold must be expanded into the corresponding five new thresholds. However, it will be apparent that the thresholds corresponding to the functions $f_2(x)$ and $f_4(x)$ are independent of the original thresholds; thus, if the original function has more than one threshold, when each is expanded the second and fourth function of each expansion will be identical and therefore pairs of these functions will cancel out. In effect, therefore, each threshold expands into only three new thresholds except for the first which expands into five new ones if the original number of thresholds is odd.

This can be summarised as follows:

Let the original function $f'(x)$ have a weight-threshold vector of

$$f'(x) = a'_1, a'_2, \ldots, a'_i, a'_j, \ldots, a'_n; t'_1, t'_2, \ldots, t'_p$$

Then (assuming all weights are positive) $f(x)$ is given by

$$f(x) \equiv a_1, a_2, \ldots, a_i, a_j, \ldots, a_n; t_1, t_2, \ldots, t_{3p+1-(-1)^p}$$

where

$$a_L = a'_L \text{ when } L = 1 \text{ to } n \quad L \neq i \neq j$$

$$a_i = \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1$$

$$a_j = \sum_{L=1}^{n} a'_L - a'_i + 1$$

and

$$t_1 = t'_1 \underbrace{- (t'_1 - \sum_{L=1}^{n} a'_L + a'_i + a'_j - 1)}$$

included if $t'_1 > \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1$

$$t_2 = \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1 \text{ included if } p \text{ is odd}$$

$$t_3 = t'_1 + \sum_{L=1}^{n} a'_L - 2a'_i - a'_j + 1 + \underbrace{(a'_i - t'_1)}$$

included if $a'_i > t'_1$

$$t_4 = 2 \sum_{L=1}^{n} a'_L - 2a'_i - a'_j + 2 \text{ included if } p \text{ is odd}$$

$$t_5 = t'_1 + 2 \sum_{L=1}^{n} a'_L - 2a'_i - 2a'_j + 2 + \underbrace{(a'_j - t'_1)}$$

included if $a'_j > t'_1$

$$t_6 = t'_2 \underbrace{- (t'_2 - \sum_{L=1}^{n} a'_L + a'_i + a'_j - 1)}$$

included if $t'_2 > \sum_{L=1}^{n} a'_L - a'_i - a'_j + 1$

$$t_7 = t'_2 + \sum_{L=1}^{n} a'_L - 2a'_L - 2a'_i - a'_j + 1 + \underbrace{(a'_i - t'_2)}$$

included if $a'_i > t'_2$

$$t_8 = t'_2 + 2 \sum_{L=1}^{n} a'_L - 2a'_i - 2a'_j + 2 + \underbrace{(a'_j - t'_2)}$$

included if $a'_j > t'_2$

t_9 to t_{11} as t_6 to t_8 but with t'_3 in place of t'_2

t_{12} to t_{10} as t_6 to t_8 but with t'_4 in place of t'_2

$t_{3p-1-(-1)^p}$ to $t_{3p+1-(-1)^p}$ as t_6 to t_8 but with t'_p in place of t'_2

Example 2 in the following Section shows the application of these arithmetic computations.

In general, functions may be resolved using both disjoint spectral translation and input spectral translation, in which case the two methods described would be used together.

Note, however, that method 1 should always be applied first since this method can only be applied to functions which initially have one threshold. When this has been done the second method can be applied to give the final solution.

It must also be noted that since the stipulation that all weights must be positive has been applied throughout, careful attention must be given to ensure that inputs with negative weights are inverted.

Example 3 in the following section illustrates these points.

2.3 *Nonthreshold kernal function*

For $n \leqslant 4$ the only nonthreshold classification of functions is the so-called 'all 4s' spectrum [4, 9], due to the fact that all its coefficients have the value of ± 4, see Table 1. The simplest of the functions covered by this classification in terms of weights and thresholds is the canonic function

$$f(x) = x_1x_2 \bullet x_1x_4 \bullet x_2x_4 \bullet x_3x_4 \bullet x_1x_3 \bullet x_2x_3$$

which has a spectrum of

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

However, in a multithreshold form it can be realised by the weights and multiple threshold values of

$$a_1, a_2, a_3, a_4; t_1, t_2$$

where

$$a_1 = a_2 = a_3 = a_4 = 1$$

$$t_1 = 2$$

$$t_2 = 4$$

All other functions in this class can now be obtained by using suitable spectral translation, i.e. Exclusive-OR's on appropriate inputs. Thus this function can be thought of as the positive canonic function covering all the 'all-4s' classified functions.

For $n > 4$ the problem of finding canonic nonthreshold functions in multithreshold form is more difficult since there would have to be a large number of them (for $n < 6$ there would have to be > 69000). It is therefore proposed that if a function is found to belong to a nonthreshold class then it should be artitrarily decomposed about one or more input variable so that each part of the decomposed function can be transformed in single or multithreshold form using the previously described methods.

3 Example: multithreshold syntheses

3.1 *Example 1*

Consider the nonlinearly-separable function

$$f(x) = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3$$

Its spectral coefficients are given in (*a*) below. In order for this function to have a threshold-logic kernal, spectral translations are required to give the magnitudes 4, 4, 4, 12 in the primary set of coefficients. Disjoint spectral translation with input x_1 and with input x_2 is therefore implemented, giving the results detailed in (*b*) and (*c*), respectively.

(*c*) is now linearly-separable, as may be confirmed from published data [1, 4]. The corresponding b_i and a_i' weights are shown in (*d*) and (*e*), giving the disjoint single-threshold realisation of Fig. 3*a*.

	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
(*a*)	4	−4	−4	−4	0	4	4	0	4	0	0	12	0	0	0	0
(*b*)	−4	4	4	4	0	−4	−4	0	12	0	0	4	0	0	0	0
(*c*)	4	−4	−4	12	0	4	4	0	4	0	0	−4	0	0	0	0
(*d*)	b_0	b_1	b_2	b_3	b_4											
	−4	−4	−4	12	0											
(*e*)	a_0'	a_1'	a_2'	a_3'	a_4'	$t' = 1$, see Fig. 3*a*										
	−1	−1	−1	2	0											

Correction of the negative a_i' weights give the modification shown in Fig. 3*b*. Finally, conversion of this realisation to the required multithreshold form is given in Fig. 3*c*, where the required multithresholds are determined using the above

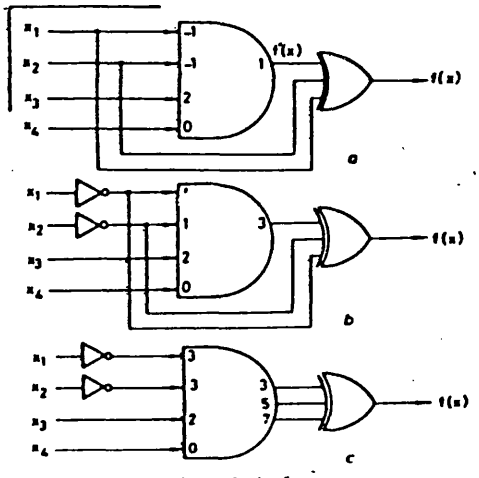


Fig. 3 *Multithreshold synthesis of*
$f(x) = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3$

a Disjoint spectral translations on given function $f(x)$
b Correction of negative input weights
c Translation of *b* into a single multithreshold realisation

developments, namely

$$c \geq t' - a_1' = 2$$

$$c \geq t' - a_2' = 2$$

$$c \geq a_2' + a_3' + a_4' + 1 - t' = 1$$

$$c \geq a_1' + a_3' + a_4' + 1 - t' = 1$$

whence $c = 2$

Therefore the final required input weights and thresholds are:

$$a_1 = 1 + 2 = 3$$

$$a_2 = 1 + 2 = 3$$

a_3, a_4 unchanged

$$t_1 = 3$$

$$t_2 = 3 + 2 = 5$$

$$t_3 = 5 + 2 = 7$$

3.2 Example 2

Consider the function

$$f(x) = x_2x_3x_4 + x_2\bar{x}_3\bar{x}_4 + \bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2x_3x_4 + x_1\bar{x}_2\bar{x}_3x_4$$

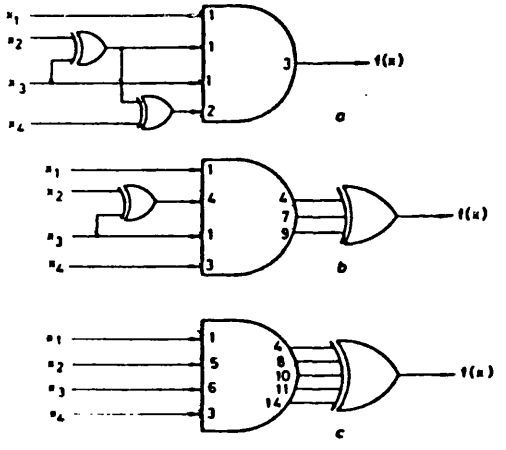


Fig. 4 *Multithreshold synthesis of*

$f(x) = x_2x_3x_4 + x_2\bar{x}_3\bar{x}_4 + \bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2x_3x_4 + x_1\bar{x}_2\bar{x}_3x_4$

a Input spectral translations on the given function $f(x)$
b Removal of the x_4 input spectral translation
c Removal of the x_3 input spectral translation

Its spectral coefficients are given in (*a*) below. Input spectral translations shown in Fig. 4*a* give the spectral coefficient revisions shown in (*b*) and (*c*), giving a single-threshold kernal in this phase of the realisation.

Elimination of these two input spectral translations by multithresholds give the two subsequent multithreshold realisations for $f(x)$, shown in Figs. 4*b* and *c*, respectively.

	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
(*a*)	0	4	0	4	0	4	0	−4	4	0	−4	0	−4	0	12	0
(*b*)	0	4	4	4	0	0	0	−4	0	12	−4	4	0	0	0	−4
(*c*)	0	4	4	4	12	0	0	0	0	0	0	4	−4	−4	−4	0

(d)

b_0	b_1	b_2	b_3	b_4
0	4	4	4	12

(e)

a_0'	a_1'	a_2'	a_3'	a_4'
0	1	1	1	2

$t' = 3$, see Fig. 4*a*

3.3 Example 3

As a final example which involves both input and disjoint spectral translation in order to achieve a threshold-logic kernal function, consider the function

$$f(x) = x_1x_3 + \bar{x}_2(\bar{x}_3 + \bar{x}_4) + x_2x_3x_4$$

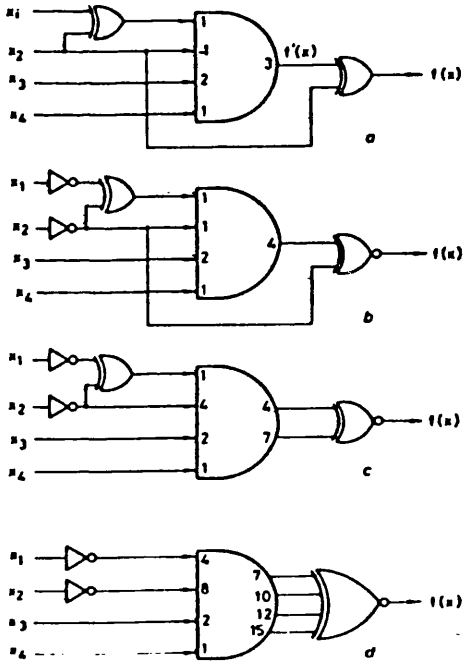


Fig. 5 *Multithreshold synthesis of*

$f(x) = x_1x_3 + \bar{x}_2(\bar{x}_3 + \bar{x}_4) + x_2x_3x_4$

a Input and disjoint translation on the given function $f(x)$
b Correction of negative input weight
c Removal of the output disjoint spectral translation
d Removal of the input spectral translation

The input spectral translation and the disjoint spectral translation to give a threshold-logic kernal are shown in Fig. 5*a*, the spectral coefficients of the given function and the result of these two translations being given in (*a*), (*b*) and (*c*) below.

	r_0	r_1	r_2	r_3	r_4	r_{12}	r_{13}	r_{14}	r_{23}	r_{24}	r_{34}	r_{123}	r_{124}	r_{134}	r_{234}	r_{1234}
(*a*)	−4	4	8	4	0	0	−4	0	8	4	0	0	4	0	−4	−4
(*b*)	8	0	−4	8	4	4	0	4	4	0	−4	−4	0	−4	0	0
(*c*)	8	4	−4	8	4	0	−4	0	4	0	−4	0	4	0	0	−4

(*d*)

b_0	b_1	b_2	b_3	b_4
8	4	−4	8	4

(*e*)

a'_0	a'_1	a'_2	a'_3	a'_4
2	1	−1	2	1

$t' = 3$, see Fig. 5*a*

Correction of the negative weighting value for a'_2 is shown in Fig. 5*b*. Elimination of the output Exclusive-OR is shown in Fig. 5*c* where c must obey

$$c \geq t' - a'_2 = 3$$

$$c \geq a'_1 + a'_3 + a'_4 + 1 - t' = 1$$

whence $c = 3$, and finally the elimination of the input spectral translation is shown implemented in Fig. 5*d*.

4 Discussion

The methods described provide a fast arithmetic means of obtaining a multithreshold solution. In practice it is often found that several alternative spectral operations may be used to synthesise a function, each method possibly giving rise to a different number of required Exclusive-OR gates. Work has been done on which is the best to select, a full treatment of which can be found in Edwards [16]. However, no work has been done on which to select to give the best set of weights and thresholds in multithreshold form, the 'best' usually considered as being the minimum integer values for the weights or the minimum number of thresholds, and hence this remains an area of further research.

The examples shown give an indication of the values of the weights and thresholds obtained using this method. In comparison with the minimised results of Haring *et al.* and Mow *et al.*, it will be found that they differ only slightly. However, this method can be applied to functions of greater than four variables.

5 Acknowledgments

This work has been pursued under UK Science Research Council Studentship 79310262.

6 References

1 DERTOUZOS, M.L.: 'Threshold logic: a synthesis approach', (M.I.T. Press, Cambridge, Mass., 1965)
2 BENNETT, L.A.M.: 'Improved forms of threshold-logic function implementation', *Electron. Lett.*, 1977, 13, pp. 368–370
3 BENNETT, L.A.M.: 'Threshold-logic functions and the Exclusive-OR', *ibid.*, 1977, 13, pp. 195–196
4 HURST, S.L.: 'Logical processing of digital signals', (Crane-Russak, N.Y., and Edward Arnold, London, 1978)
5 HURST, S.L.: 'Digital-summation threshold-logic gates: a new circuit element', *Proc. IEE*, 1973, 120, pp. 1301–1307
6 HARING, D.R.: 'Multi-threshold threshold elements', *IEEE Trans.*, 1966, EC-15, pp. 45–65
7 MONTGOMERY, J.H., and GAMBLE, H.S.: 'Basic c.c.d. logic gates', *Radio & Electron. Eng.*, 1980, 50, pp. 258–268
8 HANDY, R.J.: 'The use of CCDs in the development of digital logic', *IEEE Trans.*, 1977, 24, pp. 1049–1061
9 EDWARDS, C.R.: 'The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis', *IEEE Trans.*, 1975, C-24, pp. 48–62
10 HARING, D.R., and OHORI, D.: 'A tabular method for the synthesis of multi-threshold threshold elements', *ibid.*, 1967, EC-16, pp. 216–220
11 MOW, C.W., and FU, K.S.: 'An approach for the realization of multithreshold threshold elements', *ibid.*, 1968, C-17, pp. 32–46
12 ERCOLI, P., and MERCURIO, L.: 'Threshold logic with more than one threshold', Proceedings of 1962 IFIPS Congress Amsterdam, North Holland, 1962, pp. 741–745
13 HURST, S.L.: 'The application of multi-output threshold-logic gates to digital network design', *Proc. IEE*, 1976, 123, pp. 128–34
14 MUZIO, J.C., and HURST, S.L.: 'The computation of complete and reduced sets of orthogonal spectral coefficients for logic design and pattern recognition purposes', *Comput. and Electron. Eng.*, 1978, 5, pp. 231–249
15 LLOYD, A.M.: 'Design of multiplexer universal-logic-module networks using spectral techniques, *IEE Proc. E., Comput. and Digital Tech.*, 1980, 127, (1), pp. 31–36
16 EDWARDS, C.R.: 'Matrix methods in combinational logic design', Ph.D. thesis, University of Bath, England, 1973

HIGH-SPEED PROCESSING OF SERIAL INPUT LOGIC DATA

Indexing terms: Logic and logic design, Serial data processing

A method is presented which enables the synthesis of a serial input logic processor to be obtained quickly and simply. The synthesis gives a solution which uses modules previously used in the design of general synchronous systems.

Introduction: Recent work[1-4] has indicated a need for the efficient high-speed processing of logic data where the input and output are in a word-formatted serial stream. This is particularly relevant for data communication systems.

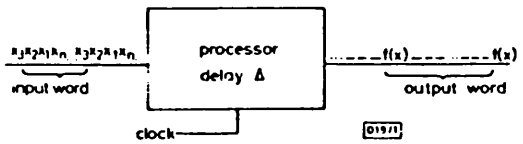


Fig. 1 *Schematic diagram of a serial logic processor*

Fig. 1 shows a schematic representation of a serial processor where a function $f(x)$ is produced from an n-bit input word after a delay of Δ clock pulses from the time of the last bit x_n entering the machine. The output word therefore consists of only one relevant bit, the remainder being 'don't care' terms.

It is desirable to include the following features in the design to maximise speed and efficiency:

(i) direct serial processing, i.e. no internal conversion from serial to parallel data
(ii) no resetting between words, thus saving a clock pulse
(iii) minimisation of the delay Δ, the minimum solution being $\Delta = 1$
(iv) minimum set of logic circuit types, e.g. modular design
(v) maximum speed modules if they are to be used
(vi) simple design strategy, e.g. utilisation of the don't care terms.

If the processor is considered as a typical synchronous machine with one input and one output, then it has been shown[5] how to obtain a realisation using the modules of Fig. 2, with a delay Δ of one clock pulse, thus satisfying conditions (i) to (iv). It has also been shown[1,6] how a high-speed bistable with a two-input variable universal-logic-module at its input can be built using ECL, which can be used as the module of Fig. 2 to satisfy condition (v). Finally it has been stated that a design procedure exists;[5] however, this procedure takes no account of the $(n-1)$ don't care terms at the output of the processor, and is consequently not an optimised solution.

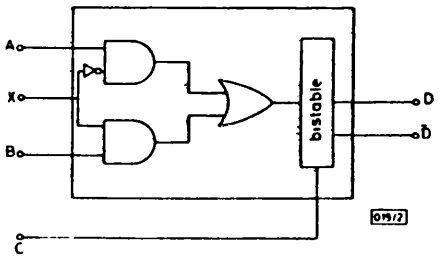


Fig. 2 *Logic module*
$D(t+1) = (\bar{X}A + XB)(t)$

However, a method has been developed[7] which does take account of the don't care terms in the general solution of synchronous circuits. This letter therefore presents a design strategy based on the above method but which relates more directly with the problem at hand. This simplifies the design procedure greatly, satisfying condition (vi).

Design procedure: A reverse response tree similar to that of Reference 7 is used, but which has a different labelling system which greatly improves the simplicity of the design. First, a node is drawn and labelled with the Boolean expression of the required function, as shown in the Example. This node then branches into two more nodes, each of which is labelled according to the decomposition of the function about the variable x_n, the left node corresponding to $x_n = 0$ and the right node corresponding to $x_n = 1$. Each of these nodes further branches, this time about the variable x_{n-1}, obeying the same rules as before. This continues until the final branching about x_1 is reached.

At each level of branching some of the nodes may be 'pruned' in accordance with the following rules:

(*a*) A node is pruned if it is labelled with a 0 or a 1.
(*b*) A node is pruned if its label f_1 is the same as the label f_2 of an unpruned node on the same level, i.e. $f_1 = f_2$.
(*c*) As (*b*) but with $f_1 = \bar{f}_2$, i.e. the inverse.

A pruned node thus becomes a terminal node, i.e. its descendants are eliminated, and its label is underlined to distinguish it from other nonterminal nodes.

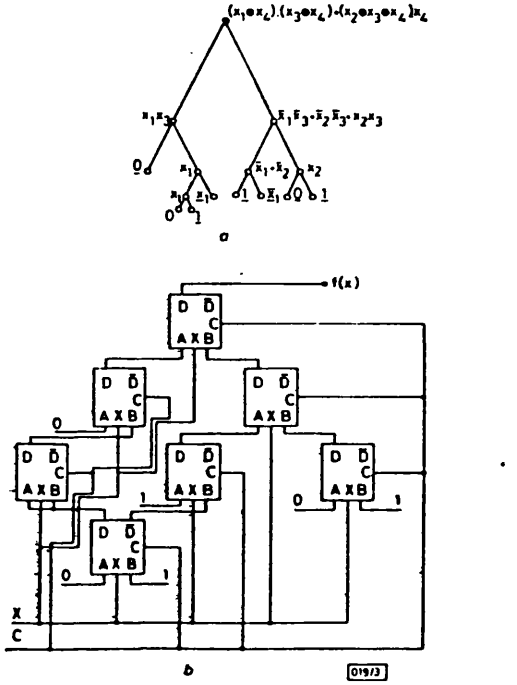


Fig. 3 *Example:* $f(x) = (x_1 \oplus x_4) \cdot (x_3 \oplus x_4) + (x_2 \oplus x_3 \oplus x_4)x_4$
a Reverse response tree
b Modular representation

Figs. 3*a* and *b* show an example of the design procedure to synthesise a particular function. It is instructive to go through the procedure step by step to ensure complete understanding.

Required function

$$f(x_1, x_2, x_3, x_4) = (x_1 \oplus x_4) \cdot (x_3 \oplus x_4) + (x_2 \oplus x_3 \oplus x_4) \cdot x_4$$
$$= x_1 x_3 \bar{x}_4 + \bar{x}_1 \bar{x}_3 x_4 + x_2 x_3 x_4 + \bar{x}_2 \bar{x}_3 x_4$$

Step 1: Decompose the function about the variable x_4:

$$f(x_1, x_2, x_3, 0) = x_1 x_3$$
$$f(x_1, x_2, x_3, 1) = x_1 x_3 + x_2 x_3 + \bar{x}_2 \bar{x}_3$$

Label the left descendant node $x_1 x_3$ and the right descendant node $x_1 x_3 + x_2 x_3 + \bar{x}_2 \bar{x}_3$.

Check whether any of rules (*a*), (*b*) or (*c*) applies to these two labels. In this instance they do not.

Step 2: Decompose about x_3:

$$f(x_1, x_2, 0, 0) = 0$$

$$f(x_1, x_2, 1, 0) = x_1$$

$$f(x_1, x_2, 0, 1) = x_1 + x_2$$

$$f(x_1, x_2, 1, 1) = x_2$$

Label the nodes from left to right accordingly.

On checking it is found that node $f(x_1, x_2, 0, 0)$ is pruned according to rule (*a*). It is therefore underlined, indicating that no further branching will occur from that node.

Step 3: Decompose about x_2:

$$f(x_1, 0, 1, 0) = x_1$$

$$f(x_1, 1, 1, 0) = x_1$$

$$f(x_1, 0, 0, 1) = 1$$

$$f(x_1, 1, 0, 1) = x_1$$

$$f(x_1, 0, 1, 1) = 0$$

$$f(x_1, 1, 1, 1) = 1$$

Label the nodes accordingly.

This time five nodes are eliminated: $f(x_1, 0, 0, 1)$, $f(x_1, 0, 1, 1)$ and $f(x_1, 1, 1, 1)$ are all pruned by rule (*a*), either $f(x_1, 0, 1, 0)$ or $f(x_1, 1, 1, 0)$ by rule (*b*) (it is immaterial which), and $f(x_1, 1, 0, 1)$ by rule (*c*). Each of the nodes is therefore underlined. Note that only nodes on the same level effect the pruning.

Step 4: Decompose about x_1:

$$f(0, 0, 1, 0) = 0$$

$$f(1, 0, 1, 0) = 1$$

Both nodes are pruned by rule (*a*): thus the branching cannot continue.

Each nonterminal node is now converted into a module. The module receives at its inputs A and B either the outputs of preceding modules in the same manner as the branches of the tree, or a logical constant 0 or 1 if the preceding node is labelled 0 or 1, respectively. If a preceding node were pruned due to rule (*c*), then the module would receive the inverse of the output of the preceding module. All modules thus have their x inputs connected to the machine input, and the design procedure is completed. This final design is illustrated in Fig. 3*b*.

Conclusions: The method described provides a fast and simple solution to the problem of serial processing, as is demonstrated in the example given. In comparison with other approaches[1-4] it shows some features which are an improvement, such as the delay time Δ and in certain cases a reduced number of modules. In fact, a useful feature is that an upper limit can be shown to be $2^n - 1$ for the number of modules that are required, where n is the number of input variables.

Finally, it is interesting to note that recent work[8,9] on the design of parallel input combinational logic employs a similar approach to this serial input logic synthesis, i.e. a tree structure and a modular realisation approach. It is therefore possible that beneficial features of either approach could be inter-related, which leads to a promising area of further research.

Acknowledgment: This work has been pursued under UK Science & Engineering Research Council scholarship 79310262.

P. D. PICTON

School of Electrical Engineering
University of Bath
Claverton Down, Bath BA2 7AY, England

7th December 1981

References

1 JONES, E. V.: 'Pipelined combinational circuits for high speed serial data', Telecommunications Group report 158, University of Essex
2 DAWS, D. C.: 'Realisation of mode controlled serial processing systems', Telecommunications Group report 159, University of Essex
3 DAWS, D. C., and JONES, E. V.: 'Hardware efficient bit sequential adders and multipliers using mode controlled logic', *Electron. Lett.*, 1980, 11, pp. 434–436
4 DAWS, D. C., and JONES, E. V.: 'Mode controlled serial logic systems', Proc. IEEE int. symp. on circuits and systems, Chicago, April 1981, pp. 902–905
5 NEWBORN, M. M.: 'A synthesis technique for binary input-binary output synchronous sequential Moore machines', *IEEE Trans.*, 1968, C-17, pp. 697–699
6 JONES, E. V.: 'High-speed transistor bistable circuits', *Electron. Lett.*, 1976, 12, pp. 375–376
7 WILLIAMS, G. H.: 'Uniform decomposition of incompletely specified sequential machines', *IEEE Trans.*, 1975, C-24, pp. 840–843
8 AKERS, S. B.: 'Binary decision diagrams', *ibid.*, 1978, C-27, pp. 509–516
9 PRICE, W. L.: 'A sequential approach to combinational logic design', *Radio & Electron. Eng.*, 1981, 51, pp. 479–504

0013-5194/82/030148-02$1.50/0

Realisation of multithreshold threshold-logic gates using charge-coupled devices

P.D. Picton, B.Sc.

Indexing terms: Logic gates, Threshold logic, CCDs

Abstract: A threshold logic gate is described which uses as its core a charge-coupled device. It is shown that the CCD is suitable for this application because of its ability to (*a*) quantise charge input to a gate, (*b*) add charge packets from different gates, and (*c*) permit conditional charge overflow to occur. The peripheral circuitry employs compatible MOSFET circuits, so that the entire gate can be made in integrated circuit form. The gate is voltage programmbale.

1 Introduction

Charge-coupled devices (CCDs) are very suitable for LSI logic design because of their high packing density, low power consumption, and their MOSFET compatibility. Previous authors [1–6] have shown that it is possible to perform binary and higher-valued logic with CCDs and actual LSI realisations have been achieved. However, the fact has been overlooked that CCDs have inherent properties which are ideally suited for the realisation of a threshold-logic element.

In the following Sections, these properties and others are discussed and a possible design for a multithreshold threshold element is given. Its main advantages over previously considered gates [7, 8] are its small size and the fact that it is voltage programmable.

2 Threshold logic

A threshold-logic gate is a device which receives binary inputs and yields a binary output, but unlike vertex gates it does not obey Boolean algebra; instead, it obeys the arithmetic summation of eqn. 1:

$$f(x) = 1 \text{ if } \sum_{i=1}^{n} a_i x_i \geq t \qquad (1)$$

$$f(x) = 0 \text{ otherwise}$$

where

x_i is a binary input
a_i is an integer weight associated with x_i
t is an integer threshold
n is the number of inputs
$f(x)$ is the binary output or function

Fig. 1 shows a typical symbol for a threshold-logic gate which can perform many but not all binary functions; those that it can being termed 'linearly separable'. In order to perform the nonlinearly separable functions, the number of thresholds in a device must be increased and each output passed through an Exclusive-OR gate as in Fig. 2. This arrangement now obeys eqn. 2:

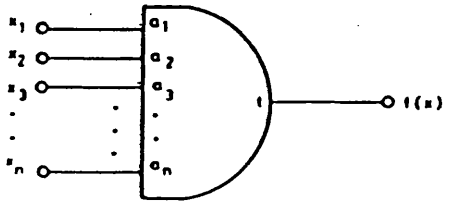


Fig. 1 General threshold logic gate symbol

Paper 1627E, first received 16th March and in revised form 27th August 1981
The author is with the School of Electrical Engineering, University of Bath, Bath BA2 7AY, England

$$f(x) = 1 \text{ if } t_{2j} \geq \sum_{i=1}^{n} a_i x_i \geq t_{2j-1},$$

$$\text{for } j = 1, 2, \ldots \qquad (2)$$

$$f(x) = 0 \text{ otherwise}$$

Thus, any binary function can be realised using a single or multithreshold threshold-logic device, providing that suitable weights and thresholds and exclusive-OR gates are available [9, 10].

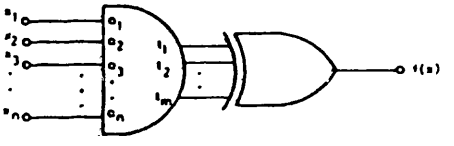


Fig. 2 General realisation of function f(x) using multithreshold threshold-logic gate and m-input exclusive-OR gate

In order that a particular device is able to perform threshold-logic operations, it must have the ability to (*a*) provide the integer-valued weights and thresholds, (*b*) sum the weights and (*c*) compare this sum with the threshold values to give the relevant 0 or 1 outputs. The following Section will now demonstrate how CCDs can provide each of these requirements.

3 CCDs [11, 12, 13, 14]

Fig. 3 shows a cross-section of a typical CCD under a single gate, where it can be seen that its structure is similar to the gate of an MOSFET. If a positive voltage V_1 is applied to the gate, then a depletion layer is created immediately beneath the gate extending into the substrate. If the gate voltage is increased, the depletion layer extends further into the substrate until a point is reached beyond which a thin layer of negative charge can be stored at the semiconductor/oxide interface, this layer of charge being equivalent to the inversion layer in a MOSFET. It is in this region that the CCD operates, the charge being used either for digital storage or as a representation of an analogue signal.

A simple model of a CCD can be used, since it is found that the surface potential decreases almost linearly as the amount of charge under the gate increases. Thus, the area immediately under the gate is known as a 'well', the charge in this well then

0143-7062/82/030107+04 $01.50/0

being analogous to a liquid, and the surface potential being analogous to the depth or distance from the 'top' of the well to the surface of the liquid. This 'depth' of the well is related to the applied gate voltage V_1, and the maximum amount of charge that can be stored in a well, known as its charge-handling capacity, is given in eqn. 3; by a first approximation this expression reduces to eqn. 4. However, if the adjacent gates have a voltage V_2 applied, as in Fig. 3, then the maximum amount of charge that can be stored under the gate is now reduced to the value given by eqn. 5. This is selfevident, since, should any more charge enter the well, it would overflow into the adjacent wells.

$$Q_{max} = C_{ox}A(V_1 - V_T) \quad (3)$$

$$Q_{max} = C_{ox}A(V_1) \quad (4)$$

$$Q_{max} = C_{ox}A(V_1 - V_2) \quad (5)$$

where

Q_{max} is the charge-handling capacity, C_{ox} oxide capacitance, A area of electrode, V_1 gate voltage, V_T threshold or inverting voltage and V_2 is the adjacent electrode voltage.

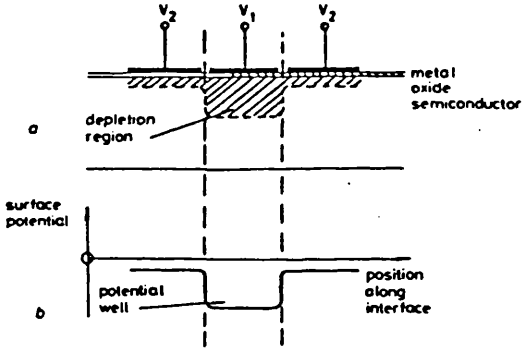


Fig. 3 *Cross-section of CCD under single electrode*

a Creation of depletion region on application of positive voltage
b Corresponding variation in surface potential showing potential well

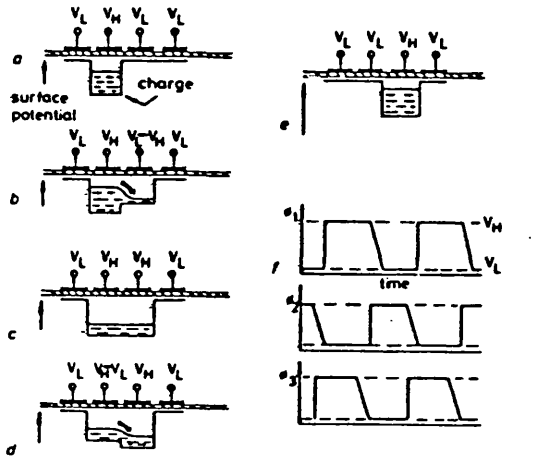


Fig. 4 *Clocked charge transfer between two electrodes*

a Charge is stored in well under ϕ_1
b and *c* ϕ_2 goes from low voltage to high voltage, and so charge flows into well under ϕ_2. The result is that the charge distributes itself equally under each electrode
d and *e* ϕ_1 goes from high voltage to low voltage, so that remaining charge under ϕ_1 flows into well under ϕ_2
f 3-phase clocking system waveforms

Charge stored in this fashion in the well can then be transferred to other wells within the device. This is usually done with a 3-phase clocking system, although 2-phase and other clocking systems are quite commonly used. Fig. 4 shows a typical transfer between two wells, and also the clock waveforms required to do this.

Some features particularly relevant to a threshold-logic device can now be discussed.

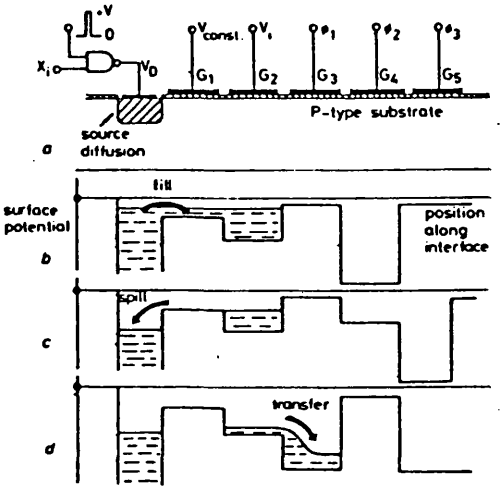


Fig. 5 *Introduction of charge to CCD*

a Cross section of CCD showing source diffusion diode and electrodes G_1 to G_5
b Surface potential under each electrode showing charge overflowing G_1 to fill G_2 when voltage on diode is pulsed to low value, i.e. when x_i is logic 1 and the other NAND gate input is pulsed to high value
c Charge spills back when voltage in diode returns to high value
d Clocked transfer begins

(a) *Charge input*

Fig. 5 shows a method of inputting a packet of charge of predetermined size. This is known as the charge equilibration or 'filled-and-spill' method in which, initially, the input diode is held at a high reverse bias, then pulsed to a low potential which causes charge to flow into the storage well G_2. When the diode is reset to its high reverse bias, excess charge flows back leaving a fixed amount in G_2; this amount is given by eqn. 6:

$$Q_i = C_{ox}A(V_i - V_{const}) \quad (6)$$

Thus a packet of charge of fixed amount can be introduced to the CCD, this amount being set by the difference of the two applied gate voltages $V_i - V_{const}$, which is equivalent ot the weight a_i in eqns. 1 and 2. In order to get the product of $a_i x_i$, a NAND gate is used with its inputs being x_i and the inverse of the pulse previously mentioned, i.e. the pulse goes from a low voltage to a high voltage. While it is at the high voltage, if the input x_i is also at a high volgage (logic 1), then the output of the NAND gate is at a low voltage and the charge packet is introduced to the system. Clearly the NAND gate can be made using MOSFET technology and therefore be able to be on the same chip as the CCD owing to their compatibility.

(b) *Charge summing*

Fig. 6 shows schematically how charge can be transferred from two wells into one common well, providing that the charge handling capacity of the common well is sufficient to hold the total amount of charge. This would also apply if more than two wells are summed.

Thus

$$Q_3 = Q_1 + Q_2 \tag{7}$$

(c) Charge overflow

Fig. 7 shows three wells. Charge is transferred from well 1 to well 2, and if the charge-handling capacity of well 2 is less than amount of charge it is presented with, there will be an overflow of charge into well 3 via the barrier gate, which is merely a gate with a low potential. Thus the charge-handling capacity of well 2, which is set by the applied gate voltage, can be considered equivalent to the threshold value in eqn. 1, and the presence or absence of charge in well 3 can be considered as an output of a logic 1 or 0, respectively. This could be extended, such that should the charge handling capacity of well 3 also be insufficient to store its presented charge; then there would be a further overflow of charge into a fourth well, and so on, thus providing the facility for multithreshold operation.

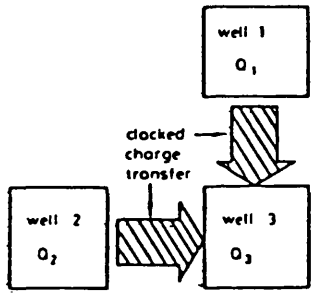


Fig. 6 *Charge summation*

4 Threshold-logic gate

Logic gates have previously been proposed, using CCDs which incorporate all of the above properties of CCDs (cf. the full adder of Zimmerman *et al.* [1]). However, these gates have been restricted to using unity charge packets at the inputs and outputs to represent the digital signal. The proposed threshold-logic gate differs from all previous gates, therefore, in that it incorporates multilevel signals owing to the weights at the input. To date, the use of multilevels in CCDs has been generally avoided and, at most, four levels have been used [2, 3, 15], although up to 32 levels have been reported [16]. The reason for this is that it is difficult to differentiate between the discrete levels at the detection part of the circuit owing to various noises that occur in CCDs [11]. However, in the proposed threshold-logic gate, the detection at the output merely looks for the presence or absence of charge in a well as explained in Section 3c. Also, the number of clocked transfers have been kept to a minimum, so as to reduce the transfer inefficiency. Thus, some of the main problems of using multilevels are overcome.

Fig. 8 shows the proposed threshold-logic gate, which has four inputs and five separate thresholds and outputs. The gate operates as follows: the four inputs are sampled at a specific point in the clock sequence and the products $a_i x_i$ formed using NAND gates as described earlier. The charge packets are then clock transferred to the large central well G5 where the charge is summed, and if this sum is greater than the charge-handling capacities of this well, charge will overflow into well G6 and possibly into wells G7 to G10. The charge-handling capacity of wells G5 to G9 are proportional to the voltage applied at their gates and are equivalent to the thresholds in eqn. 2.

The charges (if any) in wells G6 to G10 are then transferred via the output drains into the capacitors C which convert the charge into a voltage. The output circuitry is shown schematically for one output only and consists of a switch, an amplifier and a sample and hold logic circuit, the latter being required to hold the output at a logic 1 after the charge has been cleared from the capacitor via switch S to be ready for the next output signal. All this circuitry can be readily designed using MOSFETs [17], including the final stage of the gate, namely the exclusive-OR circuits [18], and thus the entire structure could be manufactured on a single IC.

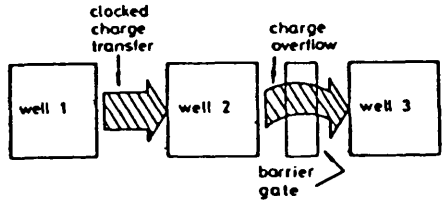


Fig. 7 *Charge overflow principle*

5 Discussion

The CCD multithreshold threshold-logic gate would seem to be a powerful gate, owing to its small size, low power consumption, MOSFET compatibility, and its programmability. However, it suffers some drawbacks owing to the inherent limitations of CCDs, such as its relatively slow speed, typically clock frequencies of a few megahertz, and the very tight tolerancing specifications. As an example of the tolerancing, consider the previous gate.

Let the transfer inefficiency ϵ be defined as the fraction of charge left behind during a clocked transfer [11, 12, 13, 14]. Thus, after the transfer from the inputs to the well G5, i.e. after four clocked transfers, the total charge entering G5 would be:

$$\sum_{i=1}^{n} a_i x_i (1-\epsilon)^4 \tag{8}$$

instead of

$$\sum_{i=1}^{n} a_i x_i, \text{ the theoretical value}$$

The worst case can be defined as being when all the inputs are at a logic 1 and the total amount of charge is equal to the highest threshold, i.e.

$$\sum_{i=1}^{n} a_i x_i = t_s \tag{9}$$

Thus, in well G10, one packet of charge should be found. The actual amount would be

$$\sum_{i=1}^{n} a_i (1-\epsilon)^4 - t_s + 1 \tag{10}$$

or

$$(1-\epsilon)^4 t_s - t_s + 1$$

This amount of charge is then transferred to the output drain, and thus the amount of charge that reaches the output is

$$O/P = (1-\epsilon)^4 \{(1-\epsilon)^4 t_s - t_s + 1\} \tag{11}$$

A typical solution of this equation, for $O/P = 0.9$, is

$$\epsilon = 0.001 \qquad t = 24$$

This states that, with a value of ϵ of 0.1%, which is reasonable for a typical CCD, then in order to get 90% of a charge packet to appear at the output, t_s which in this case equals $\sum_{i=1}^{n} a_i$ must be less than or equal to 24. Clearly, if the number of input variables increases, it usually follows that $\sum_{i=1}^{n} a_i$ increases, and thus, in order to keep a 90% charge packet at the output, the value of ϵ must decrease, which may not be possible. Therefore it would seem that a finite upper limit to the value of $\sum_{i=1}^{n} a_i$ exists.

Finally, since the threshold gate is a clocked device, it does not suffer from race hazards, and also feedback can be applied, so that the device may be used in a sequential mode.

6 Acknowledgment

This work was pursued under UK Science & Engineering Research Council Studentship 79310262.

7 References

1 ZIMMERMAN, T.A., ALLEN, R.A., and JACOBS, R.W.: 'Digital charge-coupled logic (DCCL)', *IEEE J. Solid State Circuits*, 1977, SC-12, pp. 473–485
2 KERKHOFF, H.G., and KIJKSTRA, H.: 'The application of CCD's in multiple-valued logic'. Proceedings of 5th international conference on charge coupled devices, Edinburgh, 1979, pp. 304–309
3 KERKHOFF, H.G., and TERVOERT, M.L.: 'The implementation of multiple-valued functions using charge-coupled devices'. Proceedings of 10th international symposium on multiple-valued logic, 1980, pp. 6–15
4 MONTGOMERY, J.H., and GAMBLE, H.S.: 'Basic CCD logic gates', *Radio & Electron. Eng.*, 1980, 50, pp. 258–268
5 MOK, T.D., and SALAMA, C.A.T.: 'Logic array using charge-transfer devices', *Electron. Lett.*, 1972, (8), pp. 495–496
6 HANDY, R.J.: 'Use of CCD in development of digital logic', *IEEE Trans.*, 1977, ED-24, pp. 1049–1061
7 HURST, S.L.: 'Digital-summation threshold logic gates: a new circuit element', *Proc. IEEE*, 1973, 120, pp. 1301–1307
8 BENNETT, L.A.M.: 'Improved forms of threshold-logic-function implementation', *Electron. Lett.*, 1977, 13, (12), pp. 368–370
9 HARING, D.H.: 'Multi-threshold elements', *IEEE Trans.*, 1966, EC-15, pp. 45–65
10 HURST, S.L.: 'Logical processing of digital signals' (Crane Russak, NY, and Edward Arnold, London, 1978)
11 SEQUIN, C.H., and TOMPSETT, M.F.: 'Charge transfer devices' (Academic Press, 1975)
12 HOWES, M.J., and MORGAN, D.V.: 'Charge-coupled devices and systems' (John Wiley, 1979)
13 HOBSON, G.S.: 'Charge-transfer devices' (Edward Arnold, 1978)
14 BEYNON, J.D.E., and LAMB, D.R.: 'Charge-coupled devices and their application' (McGraw-Hill, 1980)
15 YAMADA, M., FIYISHIMA, K., NAGASAWA, K., and GAMOU, Y.: 'A new multilevel storage structure for high density CCD memory', *IEEE J. Solid State Circuits*, 1978, SC-13, pp. 688–692
16 VAN ROERMUND, A., and COPPELMANS, P.: 'A circulating multilevel charge transfer device memory with adaptive refresher'. Proceedings of 5th international conference on charge coupled devices, Edinburgh, 1979, pp. 310–316
17 MAVOR, J.: 'MOST integrated-circuit engineering' (Peter Peregrinus, 1973)
18 EDWARDS, C.R.: 'Some novel exclusive-OR/NOR circuits', *Electron. Lett.*, 1975, 11, (1), pp. 3–4

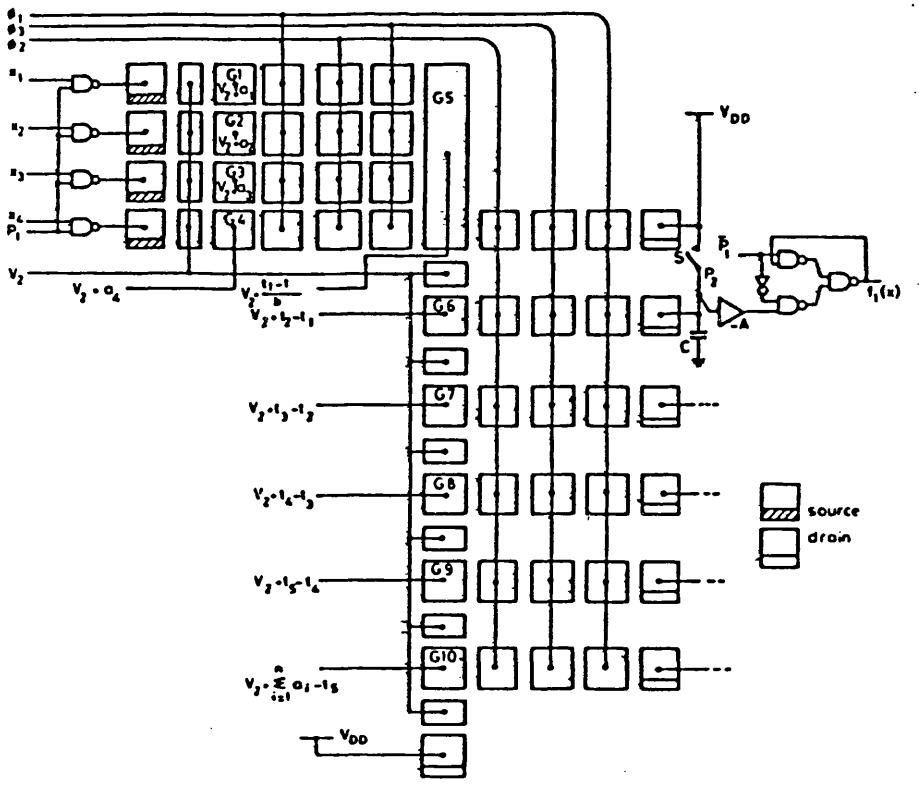


Fig. 8 *CCD multithreshold threshold-logic gate*