Logic design using digital summation threshold-logic gates

A. Pal, B.Sc., M.Tech., Ph.D., Mem.I.S.I.

Indexing terms: *Algorithms, Logic*

Abstract: The advent of IC technology has resulted in the fabrication of IC threshold gates which are competitive, both in performance and cost with standard logic packages. Of them, the multioutput digital summation threshold-logic (DSTL) gate is considered to be a potential candidate of future interest. In this paper, an algorithm has been developed to realise nonthreshold functions utilising the multioutput capability of DSTL gates. In this context, optimal realisation has been discussed. An universal logic module (ULM) has been proposed, based on DSTL approach, and an optimised structure of ULM for 4-variable functions is suggested.

1 Introduction

Threshold gate is one of the versatile complex modules proposed in the late 1950s. Since then, extensive studies have been reported on its functional properties and circuit realisation. Although threshold-logic theory flourished during the last two decades [1, 2], the threshold-gate circuit realisation based on the technology of those days faced two major problems – the circuit tolerance problem and the noise immunity problem. As a result, competitive and reliable threshold gates were not commercially available, and actual application of these gates to practical problems have been far fewer. The advent of IC technology has overcome early design problems and implementation of integrated-circuit threshold gates that are competitive both in cost and performance, with the standard logic packages, have been reported [3, 4]. Recent developments on CCD technology has also opened up the possibility of a more viable realisation of threshold-logic elements [5, 6]. This has renewed interest in this powerful module.

A novel design called digital-summation threshold-logic (DSTL) gate has been reported by Hurst [7]. A typical DSTL gate is shown in Fig. 1A. It consists of an iterative arrangement of identical cells with a uniform interconnection pattern among them. The cell details are shown in Fig. 1B. There are m input lines Y_i, $1 \leqslant i \leqslant m$ and m output lines Z_i, $1 \leqslant i \leqslant m$. The network is so connected that the output lines Z_j, $1 \leqslant j \leqslant m$, assume value 1 if any of the j number of inputs attain value 1. Thus the output line Z_j realises a threshold function with a threshold value j for the inputs Y_is, each having an input weight of 1. To provide a gate input weight greater than 1, say 2, any two such input lines may be metallised together or connected externally, providing a single external line, as shown for the input line x_1 in Fig. 1A. In this way, any input weight greater than 1 may be provided. The most important features of a DSTL gate are:

(*a*) The gate is realised by digital circuitry throughout, and so the analog tolerance problems encountered in early designs are overcome.

(*b*) It has a cellular form of realisation which is very attractive from the fabrication point of view.

(*c*) It has inherent multioutput capability which is potentially advantageous in many circumstances, e.g. as a variable threshold element, for synthesis of nonthreshold functions etc.

Although some works [8, 9] have been reported on the logic design using DSTL gates, design techniques to fully utilise the logic power of the DSTL gate are, however, still awaited. In particular, proper utilisation of the multioutput feature of the DSTL gate may lead to a reduction in overall system cost and may increase the versatility of logic design. Literature on multithreshold threshold logic elements [10–14] is relevant in this context. However, most of these works are concentrated on realisation with a minimum number of thresholds; but, because of the inherent multioutput feature of the DSTL gates the number of threshold is not a limitation. On the other hand, we shall see that the sum of weights is a more important parameter in logic design using DSTL gates.

In this paper, we have developed an algorithm where the multioutput feature of the DSTL gates has been utilised to realise nonthreshold functions. The algorithm is based on the concept of functional decomposition technique called 'isobaric decomposition'. Following this method, any nonthreshold function can be realised by using a DSTL gate in conjunction with a simple 'AND – OR' gate. In this context, optimal

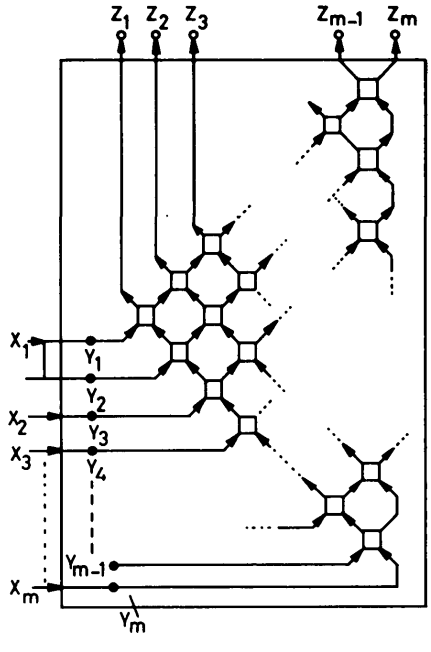

Fig. 1A *Basic digital summation threshold-logic gate*

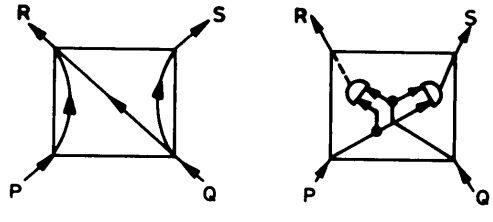

Fig. 1B *Cell details of DSTL gate*

$R = P + Q$, $S = PQ$

Paper 2249E, received 14th September 1982

The author is with the Indian Statistical Institute, 203 Barrackpore Trunk Road, Calcutta 700 035, India

realisation, a realisation with a minimum sum of weights, is discussed and a method for obtaining optimal realisation for any given function is considered. An optimised DSTL structure for realising functions of four variables has been suggested in Section 4. A commercially viable universal logic module (ULM) has been proposed in Section 5.

2 Isobaric decomposition of a function

Given a Boolean function f, the true (T) and false (F) sets of input vectors are decomposed into a mutually disjoint set of subsets:

$$F_1, F_2, \ldots, F_{k'} \text{ and } T_1, T_2, \ldots, T_k$$

(where k and k' are equal or differ by one), respectively and are arranged in a ordered manner:

$$T_k : F_{k'} : T_{k-1} : F_{k'-1} : \ldots : T_1 : F_1$$

(assume $k = k'$) such that the following conditions are satisfied:

Condition 1
The functions formed by the union of all the subsets to the right (or left) of any subset from the ordered arrangement is a threshold function, i.e. $C_i = T_k \cup F_k \cup T_{k-1} \cup \ldots \cup F_i$ (or $C_i' = T_k \cup F_k \cup \ldots \cup T_i$), $1 \leqslant i \leqslant k$, is threshold function. Let the threshold value of C_i be t_i and that of C_i' be t_i'.

Condition 2
Any pair of functions C_i and $\overset{\circ}{C}_j$, where $\overset{\circ}{C}_j$ is either C_j or C_j', $1 \leqslant i, j \leqslant k$, formed in the above manner are isobaric [2], i.e. they can be realised by threshold gates of the same weight vector but of different threshold value.

Definition 1
A decomposition of a function f satisfying the above conditions will be called the isobaric decomposition of f.

Before we proceed to obtain an algorithm for obtaining the isobaric decomposition of a given function, let us consider how this can be utilised to realise the function, using a DSTL gate.

Let the isobaric decomposition for a function f be as shown in Table 1:

Table 1: Function *f*, isobaric decomposition

T_k	F_k		T_1	F_1
.	.		.	.
.	.	. . .	.	.
.	.		.	.
t_k	t_{k-1}		t_1'	t_1

The decomposition corresponds to the weight vector $\langle w_1, w_2, \ldots, w_n \rangle$ for the n input variables, and the threshold values separating the subfunctions are shown in Table 1.

Now, consider a DSTL gate with input weight vector $\langle w_1, w_2, \ldots, w_n \rangle$ and provided with outputs $Z_1, Z_2, \ldots, Z_m$, where $m = \sum_{i=1}^{n} w_i$. Conceivably, the values of t_i and t_i' ($1 \leqslant i \leqslant k$) will lie in the range from 1 to m. From Table 1, the following expression can be readily written:

$$f_1 = T_1 + T_2 + \ldots + T_k$$
$$= Z_{t_1} \cdot \bar{Z}_{t_1'} + Z_{t_2} \cdot \bar{Z}_{t_2'} + \ldots + Z_{t_k}$$

In this way, by simple gating some of the Z_i terminals of the DSTL gate, we can realise any nonthreshold function. This is illustrated with help of the following example:

Table 2: Function of f_1, isobaric decomposition

T_3	F_3	T_2	F_2	T_1	F_1
			1		
11	9	7	8	2	0
			3		
13		14	5	4	
			10		
15			12	6	
7	6	5	3	1	

Example 1
Consider a function $f_1 = \Sigma(2, 4, 6, 7, 11, 13, 14, 15)$. The isobaric decomposition is shown in Table 2. Here $w_1 = 3$, $w_2 = 1$, $w_3 = 1$ and $w_4 = 3$.
From Table 2, we obtain $f_1 = Z_1\bar{Z}_3 + Z_5\bar{Z}_6 + Z_7$
The realisation of f_1 is shown in Fig. 2.

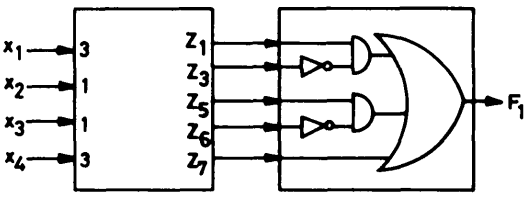

Fig. 2 *Logic network realising f_1*

3 Algorithm for obtaining isobaric decomposition

The detailed method of obtaining isobaric decomposition for a given function has been considered elsewhere [13, 16, 17]. In this paper, we shall briefly outline the method for the sake of completeness.

From the first property of isobaric decomposition, as stated in Section 2, it is evident that, for a particular weight vector, the excitation $\sum_{i=1}^{n} a_i w_i$ of any input vector in a certain subset (say T_i) should not be less than the excitation of any input vector in the succeeding subsets (T_{i-1} or F_{i-1}). This, in turn, demands that no input vector in a subset T_i (or F_i) is covered by any input vector of the succeeding subsets. A decomposition satisfying this covering property can be obtained by primary partition [13, 17]. So 'primary partition' may be considered as a first step towards isobaric decomposition. The algorithm for obtaining the 'primary partition' of a given function has been considered elsewhere [13, 17].

The second step is to check whether the functions generated by primary partition are linearly separable or not. The checking is done by testing 2-asummability [2, 17–19] of these functions. If, at any stage, the 2-asummability is not satisfied, the subsets are further decomposed, such that all the functions generated by the decomposition are linearly separable.

The second condition of isobaric decomposition requires that the functions obtained by decomposition are mutually isobaric [1]. The isobaricity of these functions are tested by using the property of mutual 2-asummability [2]. If the decomposition from the previous steps satisfies 'mutual-2-asummability' condition, the isobaric decomposition is obtained, otherwise the decomposition is further modified to satisfy mutual 2-asummability condition. The mutual 2-asummability is tested by using a technique called 'inverse mapping' [13, 16, 18].

The checking of 2-asummability and mutual 2-asummability conditions are basically table lookup methods and can be conveniently performed with the help of the Tables of Reference 18. It may be noted that 2-asummability is a necessary and sufficient condition for linear separation upto eight

variables and mutual 2-asummability is a necessary and sufficient condition, up to seven variables. Since our method is based on these properties, the algorithm is restricted to functions containing no more than seven variables.

4 Minimal realisation

It has been found that circuit realisation problems are, in general, constrained or motivated by either practical or economical, or both, reasons. In the present case, also, the synthesis technique is motivated by the following facts. From Fig. 1, it is evident that the array size of a DSTL gate depends on the value of $\sum_{i=1}^{n} w_i$ for that gate. The greater the value of $\sum_{i=1}^{n} w_i$, the larger the size of the array. The speed of the gate, in turn, depends on the array size. The larger the array size, the greater is the delay through the gate. So, if one wants to realise a function with minimum cost and with fastest response using the DSTL gate, it is necessary to synthesise the function with minimum $\sum_{i=1}^{n} w_i$. We shall call this minimal DSTL realisation. The minimal DSTL realisation, in turn, requires an isobaric decomposition of the function, corresponding to minimum $\sum_{i=1}^{n} w_i$.

It is seen that the decomposed structure obtained through isobaric decomposition may differ for transformed functions obtained by complementation of the input variables. This idea can be utilised to obtain isobaric decomposition with minimum $\sum_{i=1}^{n} w_i$. The idea is illustrated with the help of the following example:

Example 2

Consider a function $f_2 = \Sigma(0, 1, 2, 5, 6, 7, 11, 12)$. The isobaric decomposition for f_2 is shown in Table 3. Here $w_1 = 1$, $w_2 = 1$, $w_3 = 3$ and $w_4 = 2$, and so $\sum_{i=1}^{4} w_i = 7$.

Table 3: Function f_2, isobaric decomposition

F_2	T_2	F_1	T_1
13	5	4	0
14	6	8	1
15	12	3	2
	7	9	
	11	10	
6		4	2

Now, transforming f_2 by complementing the variables x_1 and x_2, we get the function $f_3 = \Sigma(3, 2, 1, 6, 5, 4, 8, 15)$. The isobaric decomposition for f_3 is shown in Table 4.

In this case $w_1 = 1$, $w_2 = 1$, $w_3 = 1$ and $w_4 = 2$ and so $\sum_{i=1}^{4} w_i = 5$. Note that the DSTL array size in this case is smaller than the previous one, but the complemented inputs $\bar{x}_1$ and $\bar{x}_2$ must be fed to the DSTL gate to realise f_2.

It is presumed that complements of the variables are available to the module inputs. To obtain the realisation for a given function, it is necessary to get isobaric decomposition for all possible complementation, which is 2^n for a function of n variables. However, it has been found that it is sufficient to consider only 2^{n-1} cases, because the nature of the isobaric decomposition and the weight-threshold vector remains unchanged if all the variables of a function are complemented.

Table 4: Function f_3, isobaric decomposition

T_2	F_2	T_1	F
15	9	1	0
	10	2	
	12	4	
	7	8	
	11	3	
	13	5	
	14	6	
5		3	1

Table 5: Equivalent classes of functions

$\sum_{i=1}^{4} w_i$	Number of equivalent classes of functions
1	1
2	2
3	4
4	13
5	14
6	38
7	41
8	53
9	8
10	59
12	4

5 Optimum DSTL gate specification

Based on the approach considered in Section 2, we have investigated the optimum DSTL gate specification, the most suitable for general-purposes usage. Results of an industrial survey shows that more than five or six binary inputs per logic area is seldom found, and an average of 3.2 binary inputs per random logic has been quoted [6]. So, we have searched for a DSTL package with minimum array size, minimum input/output connections, and one that can be used to realise all (or most of the) equivalent classes of functions of four variables. Minimal DSTL realisations for all equivalent classes of functions of four variables were obtained with the help of computer. A summary of the results is shown in Table 5, which shows that, out of the 237 equivalent classes of functions, only four equivalent classes require $\sum_{i=1}^{n} w_1 = 12$, and the rest of the equivalent classes can be realised with $\sum_{i=1}^{4} w_i \leqslant 10$. To realise all possible functions with minimal isobaric decomposition corresponding to $\sum_{i=1}^{4} w_i \leqslant 10$, the possible DSTL structure is shown in Fig. 3. The gate has specification $\langle 2, 2, 2, 1, 1, 1, 1\rangle$ and all the ten output Z_1 to Z_{10} are available. We shall call this optimised DSTL package for $n = 4$. It can be accommodated in a 20-pin dual-in-line package (DIP).

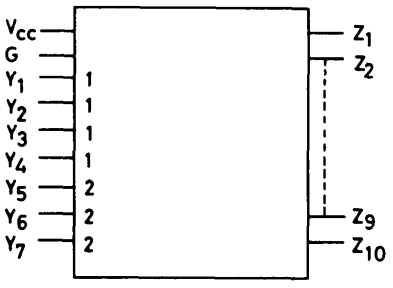

Fig. 3 *Optimised DSTL package*

The AND–OR package, which can be used in conjunction with the optimised DSTL package to realise nonthreshold functions, is shown in Fig. 4.

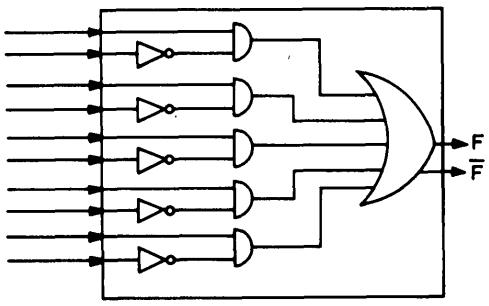

Fig. 4 *AND – OR package*

From Table 5 we find that there are four equivalent classes of functions of four variables which require $\Sigma w_i = 12$ for their realisation. These functions cannot be realised using the optimal DSTL package considered above. A function of this type can be realised by decomposing the function about one of the input variables. This is illustrated by the following example:

Example 3
Consider a function $f_3 = \Sigma(0, 1, 2, 3, 4, 10, 13)$. By expanding about x_4, we obtain

$$f_3 = x_4 f_3' + \bar{x}_4 f_3''$$

where

$$f_3' = \Sigma(2, 5) \text{ and } f_3'' = \Sigma(0, 1, 2, 3, 4)$$

The isobaric decomposition of f_3' and f_3'' is shown in Tables 6 and 7, respectively. The realisation of f_3 is shown in Fig. 5. The above technique can also be used to realise functions of five variables.

Table 6: Function f_3', isobaric decomposition ($w_1 = w_3 = 1, w_2 = 2$)

F_2	T_1	F_1
3	2	0
		1
6	5	2
7		4
3	1	

Table 7: Function f_3'', isobaric decomposition ($w_1 = w_2 = 1, w_3 = 2$)

F_1	T_1
5	0
	1
6	2
7	4
	3
3	

6 Universal logic module

In a conventional logic module (ULM), the function selection is done by changing the input/output terminal connection while the internal structure remains invariant, whereas, in cellular logic arrays, the input/output connections remain invariant while the cell function is changed to realise different functions [22]. Utilising both these concepts, we have formulated a technique for the realisation of universal logic module for functions of four variables.

In realising a function using a DSTL gate, it is found that the method essentially consists of two steps: (*a*) proper interconnection of the input terminals to realise a particular input weight vector, and (*b*) proper interconnection of the Z_i terminals to the inputs of the AND–OR gate. As discussed in Section 5, functions of four variables can be realised using the optimised DSTL gate of Fig. 3 and the AND–OR package of Fig. 4. Now consider the case where both of these circuits are fabricated on a single chip. The input terminals of DSTL gate and output of AND–OR gate are externally provided, whereas interconnections among the DSTL gate outputs and the AND–OR gate inputs are internal by a metallisation technique.

Thus, by realising functions in two stages, one at the time of fabrication and the other at the time of practical implementation, this provides us with a DSTL-based universal logic module.

7 Conclusions

In this paper, an algorithm has been presented, realising nonthreshold functions utilising the multioutput feature of DSTL gates. It is simple, straightforward and basically a table-lookup method. The Tables in Reference 18 can be conveniently utilised, and so it does not involve much computation.

It may be noted that the realisation considered here is similar to the synthesis of multithreshold threshold elements [10–14]. Hence, it is not irrelevant to compare results obtained by previous workers in this field; particularly Haring and Ohori [10] and Mow and Fu [11] have provided the multithreshold threshold realisation of all 221 equivalence classes of functions of four variables in tabular form. By looking at the Tables in References 10 and 11, it is found that there are many equivalence classes of functions which require a sum of weights greater than ten in the multithreshold threshold realisations. Table 8 gives the number of equivalent classes requiring sum of weights greater than ten. However, the main objective of Haring and Ohori [10] and Mow and Fu [11] was to obtain realisations with a minimum number of thresholds.

Recently, Picton [14] has reported the realisation of multithreshold threshold logic networks, using the Rademchar-Walsh transform. This is an extension of the spectral translation methods of Edwards [15]. Although this method does not involve much computation, it usually leads to a nonminimal

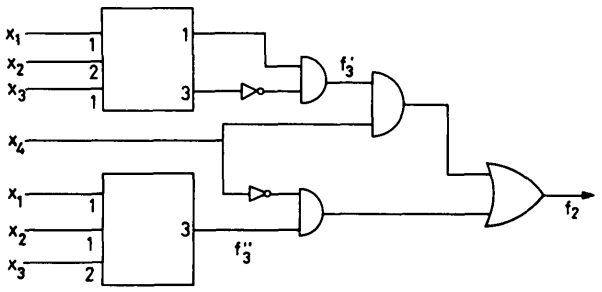

Fig. 5 *Realisation of f_3*

Table 8: Equivalent classes of functions

	Number of equivalent classes of functions		
Σw_i	Haring and Oheri [10]	Mow and Fu [11]	Present method
11	19	11	–
12	17	10	4
13	11	3	–
14	10	6	–
15	2	–	–
16	6	1	–
17	1	–	–
18	2	1	–

realisation in terms of sum of weights or the number of thresholds. For example, consider the examples in Reference 14.

Example 4
Let

$$f_4 = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3 = \Sigma(1, 2, 4)$$

Picton's realisation (see example 1) needs $\Sigma w_i = 8$ and three threshold, whereas in our method it needs $\Sigma w_i = 3$ and two thresholds. The realisation of f_4 is shown in Fig. 5.

Example 5
Let

$$f_5 = x_1x_3 + \bar{x}_2(\bar{x}_3 + \bar{x}_4) + x_2x_3x_4$$
$$= \Sigma(0, 1, 2, 8, 9, 10, 7, 11, 14, 15)$$

Picton's realisation (see example 3) needs $\Sigma w_i = 15$ and four thresholds.

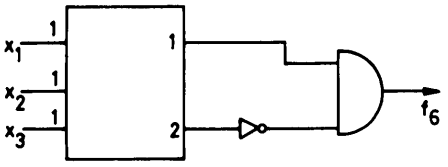

Fig. 6 *Realisation of f_4*

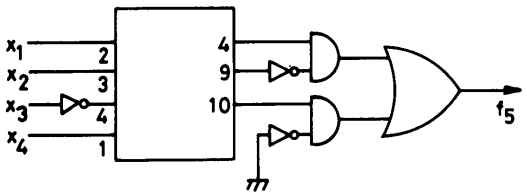

Fig. 7 *Realisation of f_5*

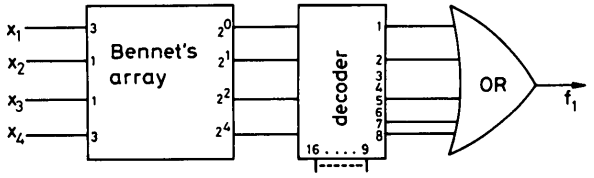

Fig. 8 *Realisation of f_1 using Bennet's array*

The realisation of this function by our method is shown in Fig. 6. Here it needs $\Sigma w_i = 10$ and three thresholds.

The improved forms of threshold-logic function implementation proposed by Bennet [20, 21] can be used in the realisation of a function as considered in this paper. After obtaining the isobaric decomposition for a given function, it can be realised by following the method discussed by Bennet [21]. Here, Bennet's array outputs should be used in conjunction with a decoder and a OR gate. For example, the realisation of f_1 using Bennet's array is shown in Fig. 8. The ULM considered in Section 6 can also be implemented using Bennet's array. In this case, Bennet's array, the decoder and the OR gate can be fabricated on a single chip, but the interconnections, depending on the function to be realised.

8 Acknowledgment

The author wishes to thank the unknown referees for their constructive suggestions.

9 References

1 SHENG, C.L.: 'Threshold logic' (Academic Press, New York, 1969), 206 pp
2 MUROGA, S.: 'Threshold logic and its applications' (Wiley-Interscience, New York, 1971), 478 pp
3 HAMPEL, D.: 'Multifunction threshold gates', *IEEE Trans.*, 1973, C-22, pp. 197–203
4 HAMPEL, D., and WINDER, R.O.: 'Threshold logic', *IEEE Spectrum*, 1971, 8, pp. 32–39
5 HANDY, R.J.: 'The use of CCD's in the development of digital logic', *IEEE Trans.*, 1977, EC-16, pp. 1049–1061
6 MONTGOMERY, J.H., and GAMBLE, H.S.: 'Basic CCD logic gates', *Radio & Electron. Eng.*, 1980, pp. 258–268
7 HURST, S.L.: 'Digital summation threshold logic gates: a new circuit element', *Proc. IEE*, 1973, **120**, (11), pp. 1301–1307
8 HURST, S.L.: 'Logic network synthesis using digital summation threshold logic gates'. Proceedings of twelfth annual Allerton conference on circuit and system theory, Oct. 1974, pp. 525–534
9 HURST, S.L.: 'Application of multioutput threshold logic gates to digital network design', *Proc. IEE*, 1976, **123**, (2), pp. 128–134
10 HARING, D.R., and OHORI, D.: 'A tabular method for the synthesis of multithreshold elements', *IEEE Trans.*, 1967, EC-16, pp. 216–220
11 MOW, C.W., and FU, K.S.: 'An approach for the realization of multithreshold threshold elements', *ibid.*, 1978, C-17, pp. 32–46
12 HARING, D.R., and DIEPHUIS, R.J.: 'A realization procedure for multithreshold threshold elements', *ibid.*, 1967, EC-16, pp. 828–835
13 GHOSH, S., and CHOUDHURY, A.K.: 'Partition of Boolean functions for realization with multi-threshold threshold elements', *ibid.*, 1973, C-22, pp. 204–215
14 PICTON, P.D.: 'Realization of multithreshold threshold logic network using the Rademchar-Walsh transform', *IEE Proc E, Electron. Circuits & Syst.*, 1981, **128**, (3), pp. 107–113
15 EDWARDS, C.R.: 'The application of the Rademchar-Walsh transform to Boolean function classification and threshold logic synthesis', *IEEE Trans.*, 1975, C-24, pp. 48–62
16 PAL, A.: 'Studies on the synthesis of digital logic circuits using threshold gates and MOS modules'. Ph.D. thesis, University of Calcutta, 1975
17 DE, P., SEN, A., PAL, A., SARMA, D., and CHOUDHURY, A.K.: 'Minimal realization of arbitrary switching functions with 2-level network of isodistinct and isobaric threshold gates', *Int. J. Syst. Sci.*, 1974, 5, pp. 555–573
18 DE, P., PAL, A., SEN, A., SARMA, D., and CHOUDHURY, A.K.: 'A tabular method for finding 2-summable and mutually 2-summable pairs', *Int. J. Electron.*, 1976, 37, pp. 409–427
19 PAL, A.: 'An iterative algorithm for testing 2-asummability of Boolean functions', *Proc. IEEE*, 1981, 69, pp. 1164–1166
20 BENNET, L.A.M.: 'Improved forms of threshold-logic-function implementation', *Electron Lett.*, 1977, 13, (12), pp. 368–370
21 BENNET, L.A.M.: 'Threshold logic functions and the exclusive-OR', *ibid.*, 1977, 13, (7), pp. 195–196
22 STONE, S.H.: 'Universal logic modules', *in* 'Recent development in switching theory', MUKHOPADHYAY, A. (Ed.): (Academic Press, New York, 1971), pp. 229–254

Ajit Pal was born in 1949 in West Bengal, India. He received the B.Sc., M.Tech. and Ph.D. degrees from Calcutta University in 1968, 1971 and 1976, respectively.

From 1972 to 1975 he was research fellow of the Department of Atomic Energy and University Grants Commission, Government of India. In 1976, he served for a brief period – about four months – at the Radar Division of the Defence Electronics Research Laboratory, Hyderabad. In the same year he joined Indian Telephone Industries Ltd., Naini, where he has been concerned with the design and development of a number of digital communication and test equipments. Since May 1979 he has been on the faculty of the Indian Statistical Institute, Calcutta. His current interests include microprocessor-based systems, logic design, automata theory and fault tolerance computing.

Dr. Pal is a member of the Indian Statistical Institute, India.