A New Synthesis of Symmetric Functions

Hafizur Rahaman
Computer Sc. & Tech.
A. P. C. Roy Polytechnic College
Kolkata –700 032, India
rahaman_h@hotmail.com

Debesh K. Das
Computer Sc. & Engg.
Jadavpur University
Kolkata –700 032, India
debeshd@hotmail.com

Bhargab B. Bhattacharya
Computer Sc. & Engg.
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
bhargab@cse.unl.edu

Abstract

A new approach to synthesizing totally symmetric Boolean functions is presented. First, a novel cellular array is introduced for synthesizing unate symmetric functions. Using this module, a general symmetric function is then realized following a unate decomposition method. The cellular structure is simple and universal – it uses only 2-input, 2-output AND-OR cells, and admits a recursive construction. The design provides a significant reduction in hardware cost compared to other existing techniques.

1. Introduction

Synthesis of symmetric Boolean functions is a classical problem in switching theory, and several techniques are well known [1-4, 9]. Since symmetric functions play a key role in cryptology [5], design of easily testable circuits to realize them has received considerable attention recently. This paper presents a new approach to synthesizing totally symmetric functions. We first introduce a novel cellular logic array for realizing unate symmetric functions. These modules are constructed using a simple iterative arrangement of 2-input, 2-output AND-OR cells. A similar network known as *digital summation threshold logic* (DSTL) array was reported earlier by Hurst [7] in connection to threshold logic. Testable logic design using DSTL arrays for detecting stuck-at and bridging faults appeared in [6]. In this work, we present a new and compact cellular structure that realizes the same set of logic functions as that of a DSTL array with significant reduction in hardware cost and delay. Following a unate decomposition technique, we show that any totally symmetric function can be synthesized using these modules. Our design approach is universally applicable to any general symmetric function and hardware cost reduces drastically compared to other existing designs [2, 3].

2. Preliminaries

A Boolean function is called *unate*, if each variable appears either in complemented or uncomplemented form (but not both) in its minimum sum-of-products (s-o-p) expression. A function *is positive (negative) unate* if each variable appears in complemented (uncomplemented) form in its minimum s-o-p. A *vertex (minterm)* is a set of variables in which every variable appears once. The *weight* w of a vertex v is the number of uncomplemented variables appearing in v.

A switching function $f(x_1, x_2, ..., x_n)$ is called *totally symmetric* with respect to the variables $(x_1, x_2, ..., x_n)$, if it is invariant under any permutation of the variables [4]. Total symmetry can be specified by a set of integers (called a-numbers) $A = (a_i,.., a_j,\ldots, a_k)$, where $A \subseteq \{0, 1, 2,\ldots, n\}$; all the vertices with weight $w \in A$ will appear as true minterms in the function. Henceforth, by a symmetric function, we would mean a function with total symmetry. An n-variable symmetric function is denoted as $S^n(a_i,..., a_j, .., a_k)$. A symmetric function is called *consecutive*, if the set A consists of only consecutive integers $(a_l, a_{l+1},..., a_r)$. Such a consecutive symmetric function is expressed by $S^n(a_l - a_r)$ where $l < r$. For n variables, we can construct 2^{n+1}-2 different symmetric functions (excluding constant functions 0 and 1). A totally symmetric function $S^n(A)$ can be expressed uniquely as a union of maximal consecutive symmetric functions, such that $S^n(A) = S^n(A_1) + S^n(A_2) +\ldots\ldots\ldots + S^n(A_m)$, such that m is minimum and $\forall$ i , j , $1 \le i, j \le m$, $A_i \cap A_j = \varnothing$, whenever $i \ne j$.

Example 1: The symmetric function $S^{12}(1,2,5,6,7,9,10)$ can be expressed as $S^{12}(1\text{-}2)+ S^{12}(5\text{-}7)+ S^{12}(9\text{-}10)$, where $S^{12}(1\text{-}2)$, $S^{12}(5\text{-}7)$ and $S^{12}(9\text{-}10)$ are maximal consecutive symmetric functions.

A function is called *unate symmetric* if it is both unate and symmetric. It can be shown that a unate symmetric function is always consecutive and can be expressed as $S^n(a_l\text{-}a_r)$, where either $a_l = 0$ or $a_r = n$. If it is positive unate, then it must be either $S^n(n)$ or any of the following (n-1) functions: $S^n(1\text{-}n)$, $S^n(2\text{-}n)$, $S^n(3\text{-}n),\ldots.$, $S^n((n\text{-}1) - n)$. We express $S^n(n)$ as $u_n(n)$, and $S^n(a_l\text{-}a_r)$ as $u_l(n)$ for $1 \le l \le (n\text{-}1)$.

Theorem 1[3]: A consecutive symmetric function $S^n(a_l\text{-}a_r)$, $a_l \ne a_r$, $l < r$, can be expressed as a composition of two unate and consecutive symmetric functions as follows:

(i) $S^n(a_l\text{-}a_r) = S^n(a_l\text{-}a_n)\ \overline{S}^n(a_{r+1}\text{-}a_n)$

(ii) $S^n(a_l\text{-}a_r) = \overline{S}^n(0\text{-}a_{l-1})\ \overline{S}^n(a_{r+1}\text{-}a_n)$

(iii) $S^n(a_l\text{-}a_r) = S^n(0\text{-}a_r)\ \overline{S}^n(0\text{-}a_{l-1})$

(iv) $S^n(a_l\text{-}a_r) = S^n(0\text{-}a_r)\ S^n(a_l\text{-}a_n)$.

3. DSTL array

The DSTL array proposed by Hurst [7] is an n-input and n-output cellular array, consisting of an iterative arrangement of identical cells having a uniform interconnection pattern among them. The design is shown in the Fig. 1a, where each cell consists of a two-input AND gate and an OR gate as shown in Fig. 1b. There are n-inputs lines x_1, x_2, x_3,….., x_n, and n output lines $u_1(n)$, $u_2(n)$, $u_3(n)$,….., $u_n(n)$ in the array. Each output u_i, implements a unate symmetric function as described below:

$$u_1(n) = S^n(1, 2, 3,…., n) = x_1 + x_2 + x_3 + \ldots\ldots\ldots\ldots + x_n$$

$$u_2(n) = S^n(2, 3, 4….., n) = x_1x_2 + x_1x_3 + \ldots\ldots\ldots\ldots + x_{n-1}\,x_n$$

$$u_3(n) = S^n(3, 4,…, n) = x_1x_2x_3 + x_1x_2x_4 + \ldots\ldots\ldots + x_{n-2}\,x_{n-1}x_n$$

$$\vdots \qquad\qquad \vdots$$

$$u_n(n) = S^n(n) = x_1x_2\ldots\ldots x_{n-1}\,x_n$$

4. Proposed technique

4.1 Synthesis for unate symmetric functions

We first introduce a basic logic module, which is similar to DSTL array in functionality, but more compact in structure.

4.1.1 Basic module

Our basic component *Module(n)* is a logic block with n input lines and n output lines (Fig. 2a). It consists of an iterative arrangement of cells, where each cell consists of a two-input AND gate and a two-input OR gate as in a DSTL array [7]. For ease of representation, we redraw the cell as shown in Fig. 2b. Though we use the same cells as in [7], our design differs significantly from the DSTL structure as far as the interconnections of cells are concerned.

Example 2: For n = 4, the DSTL array and the proposed logic module are shown in Figs. 3a and 3b. The new design needs fewer cells and has less delay compared to the DSTL array.

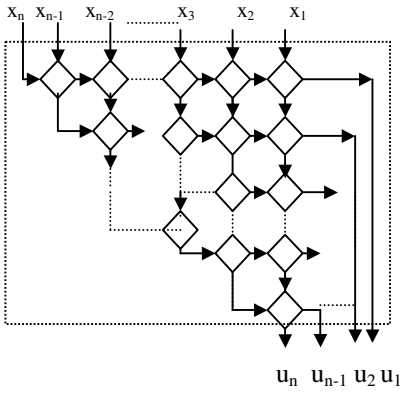

Fig. 1a: Basic DSTL array

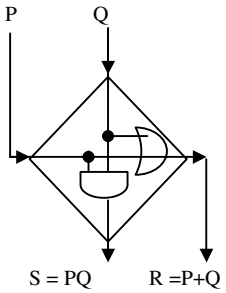

Fig. 1b: A DSTL cell

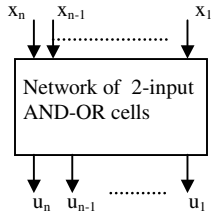

Fig. 2a: Module(n)

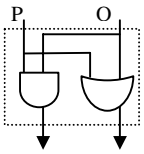

Fig. 2b: AND-OR cell

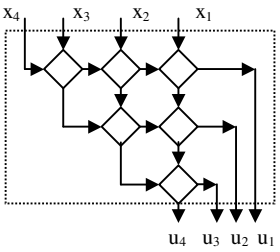

Fig. 3a: 4-variable DSTL array

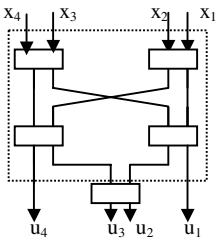

Fig 3b: 4-variable proposed array

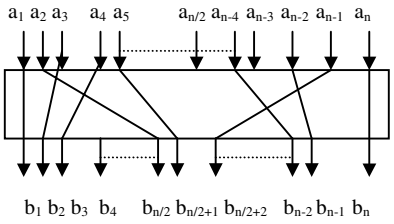

Fig. 4: Interconnect(n)

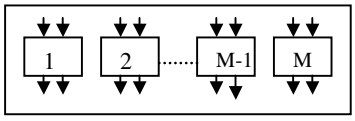

Fig. 5: Cell-cluster(M)

4.1.2 Basic structures

We use the following basic structures for our design.

Interconnect(n): Let n = even. The module Interconnect(n) provides a one-to-one and onto connection from n inputs a_1, a_2, a_3,…., a_n to n outputs b_1, b_2, b_3,…., b_n. The mapping, which is analogous to shuffle-exchange, can be expressed as

(i) for $i \leq n/2$, $b_i = a_{2i-1}$

and (ii) for $i > n/2$, $b_i = a_{i-n/2}$

An example is shown in Fig. 4.

Cell-Cluster(M): This consists of M cells in parallel (see Fig. 5), where each cell is shown in Fig. 2b. The value of M may be even or odd.

Connection[n: 1-to-2] : This uses Interconnect(n-4), and is shown in Fig. 6.

Connection[n: 2-to-3]: This depends on the number of variables. The outputs of stage-2 are divided into two parts, each having (n/2) inputs. The corresponding outputs of each partition are paired together.

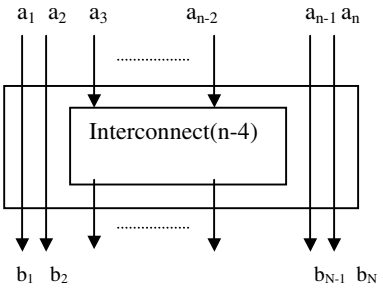

Fig. 6: Connection[n: 1-to-2]

4.1.3 Complete module (for n = even)

The details of Module(n) are shown in Fig. 7. It consists of three stages, each having n inputs and n outputs.

First-stage: This is first defined for n = even. It consists of $\lceil \log n \rceil$ levels as shown in Fig. 8. Each level consists of several cells in parallel.

Example 3: The first-stage for n = 8 and 6 are shown in Figs. 9a and 9b respectively.

For n = odd, we first design the first-stage for (n+1) inputs. Then, we set one input variable to logic 0 and remove the affected logic cells from the circuit level by level.

Example 4: The first-stage for n =7 is shown in Fig. 10a. An equivalent realization is shown in Fig. 10b.

Hardware requirement: For n = even, the number of cells in the first-stage circuit is $n/2\lceil \log n \rceil$.

Second-stage: The second stage consists of two parts, each implemented with a Module((n-2)/2). It is shown Fig. 11. Outputs of this stage are connected to the third-stage.

Third-stage: The third stage consists of a cascade of cells whose strucure is shown in Fig. 12.

Example 5: The overall designs for Module(6) and Module(8) are shown in Figs. 13a and 13b.

Example 6: Module(16) is shown in Fig. 14.

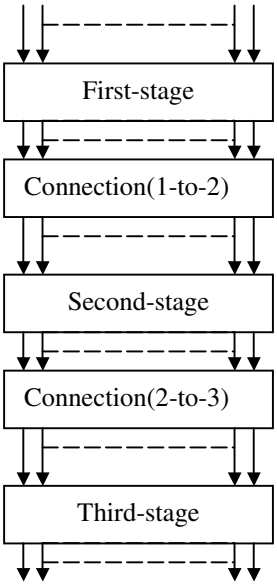

Fig. 7: Structure of Module(n)

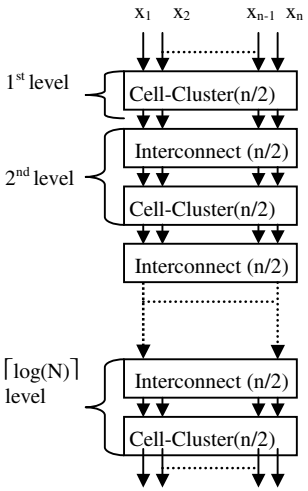

Fig. 8: First-stage of Module(n)

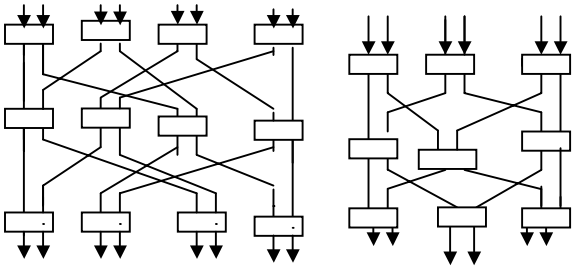

Fig. 9a: First-stage for n = 8 Fig. 9b: First-stage for n = 6

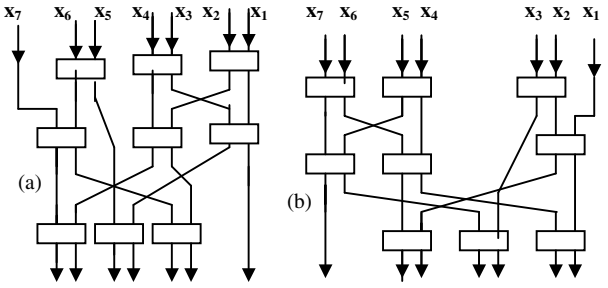

Fig.10: Two realizations of first-stage for n = 7

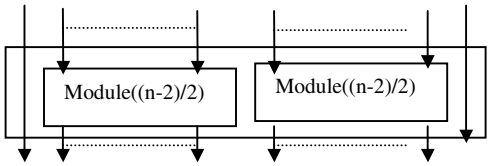

Fig. 11: Second-stage

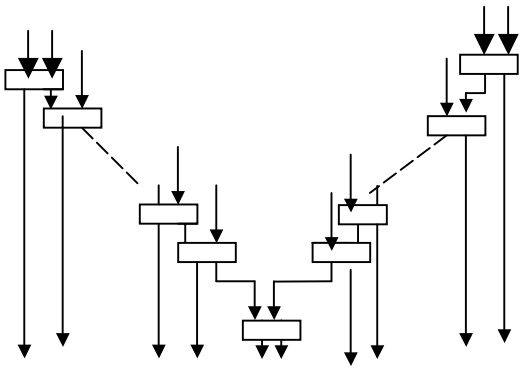

Fig. 12 Third-stage

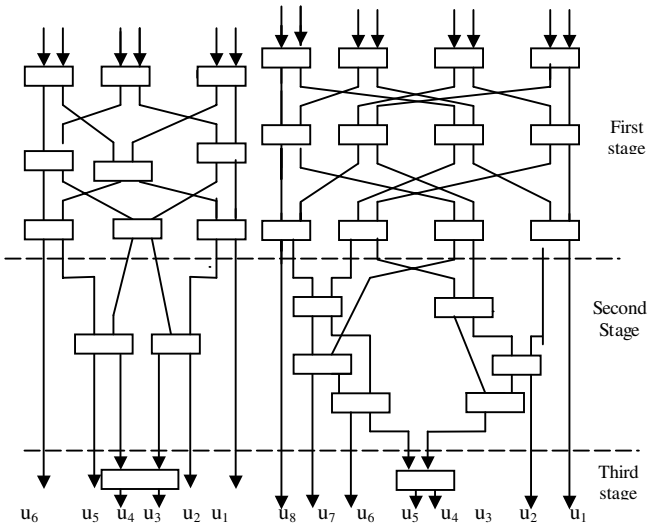

Fig. 13a: Module(6) Fig. 13b: Module(8)

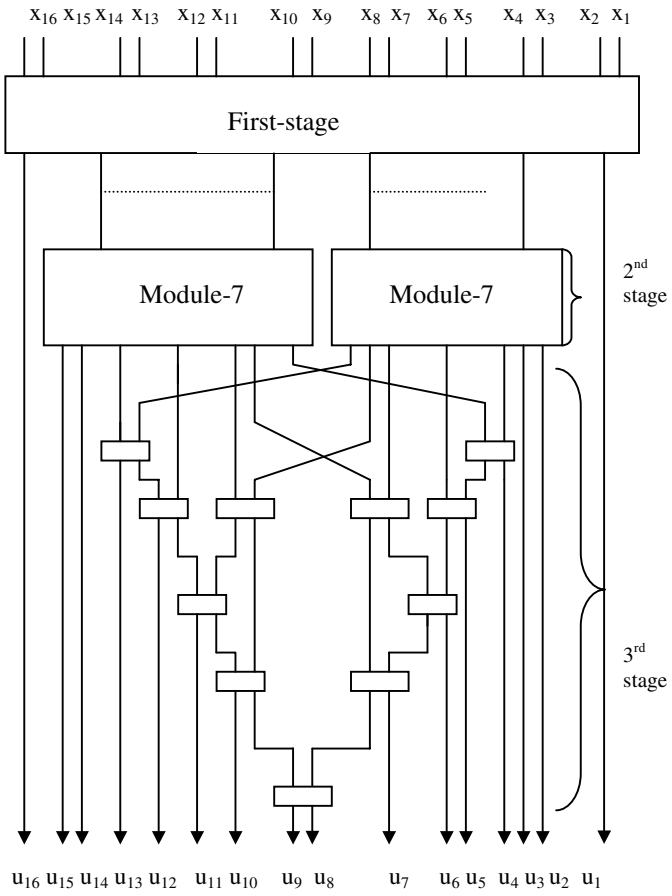

Fig. 14: Module(16)

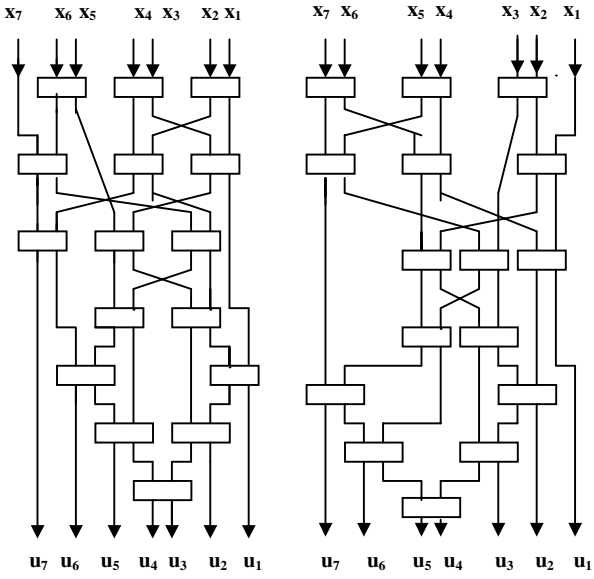

Fig. 15a: Module(7)

Fig. 15b: Module(7)

4.1.4 Designing for odd values of n

In this case, we first design the Module(n+1). Then, we set one variable to logic 0 and remove the affected logic cells in Module(n+1) to realize Module(n). These logic cells are removed only from the 1st stage.

Example 7: Module(7) is obtained from Module(8) of Fig. 13b by removing one variable and some cells. We can realize it in two ways as shown in Figs. 15a and 15b.

4.1.5 Hardware cost and delay

Let C(n) denote the number of 2-input cells in Module(n). Then $C(n) \leq n\log n + 2C(n/2) + O(n)$.
Hence, $C(n) = O(n\log^2 n)$.

Circuit delay

We assume unit gate delay through a 2-input gate. For an n-input function, the minimum delay through the circuit is $\lceil \log(n) \rceil$, and for or $n = 2^k$, the maximum delay is (n - 1).

4.2 Synthesis of general symmetric functions

To synthesize a consecutive symmetric function which is not unate, we use the result stated in Theorem 1 that $S^n(a_l - a_r)$ can be expressed as a composition of two unate symmetric functions.

Hence, $S^n(a_l\text{-}a_r) = u_l(n)\ \overline{u}_{r+1}(n)$.

The unate functions $u_l(n)$ and $u_{r+1}(n)$ are produced by the Module(n). The complete circuit is shown in Fig. 16a.

Example 9: $S^6(3,4)$ is realized as $S^6(3,4) = S^6(3\text{-}6)\ \overline{S}^6(5\text{-}6)$ $= u_3(3)\ \overline{u}_5(5)$. The circuit is shown in Fig. 16b.

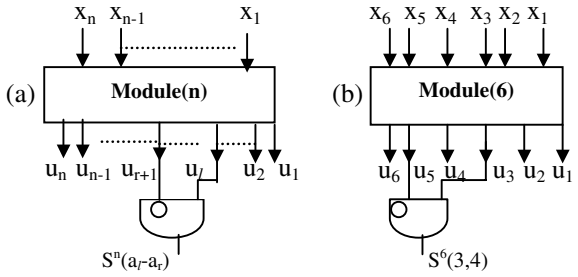

Fig. 16: Realization of (a) $S^n(a_l - a_r)$ (b) $S^n(3,4)$

Since a non-consecutive symmetric function $S^n(A)$ can be expressed uniquely as a union of maximal consecutive symmetric functions, a general symmetric function can be realized by OR-ing together the outputs of constituent consecutive symmetric functions.

5. Experimental results

We compare the hardware cost and delay of the proposed design with a DSTL array [7]. For an n-input DSTL array, one has the following parameters:

(i) number of cells = n(n-1)/2;
(ii) minimum delay = n;
(iii) average delay = 0.5(3n-1);
(iv) maximum delay =2n - 1.

Table 1 shows that the number of cells (where each cell consists of two two-input gates) and circuit delay for our design compared to those in [7].

Table 1: Cost and delay for realizing unate symmetric functions

n	# cells		delay					
	As in [7]	Proposed Method	As in [7]			Proposed Method		
			Min	Avg	Max	Min	Avg	Max
2	1	1	2	2.5	3	1	1	1
3	3	3	3	4	5	2	2.67	3
4	6	5	4	5.5	7	2	2.5	3
5	10	9	5	7	9	3	4.2	5
6	15	12	6	8.5	11	3	4	5
7	21	16	7	10	13	3	5.57	7
8	28	19	8	11.5	15	3	5.25	7
9	36	29	9	13	17	4	10.44	10
10	45	32	10	14.5	19	4	9.8	10
11	55	43	11	16	21	4	9.09	12
12	66	47	12	17.5	23	4	8.67	12
13	78	54	13	19	25	4	9.69	13
14	91	58	14	20.5	27	4	9.28	13
15	105	71	15	22	29	4	11.07	15
16	120	75	16	23.5	31	4	10.63	15

For general consecutive symmetric functions, we compare hardware cost with those in [2] and [3] in terms of the number of gate inputs (Table 2). The results show a drastic reduction in cost. While these earlier methods use fixed number of logic levels, for instance, at most 4 [2], or at most 5 [3], the proposed method reduces logic significantly at the cost of increasing the number of logic levels.

6. Conclusion

We have introduced a new technique for synthesizing a symmetric function using a cellular structure. The proposed module for realizing unate symmetric functions needs less hardware and delay compared to a DSTL array. For general symmetric functions, our synthesis method based on unate decomposition yields very low cost circuits compared to earlier methods [2, 3]. Testability issues of this design, and extending the technique for synthesizing an arbitrary Boolean function by multi-threshold partitioning [8], will be reported in a future work.

Table 2: Cost of general symmetric functions

Functions	Number of gate inputs		
$S^n(a_l\text{-}a_r)$	As in [2]	As in [3]	Proposed Method
$S^5(3,4)$	47	32	38
$S^5(2,3)$	50	38	38
$S^5(1,2)$	47	32	38
$S^6(4,5)$	83	56	50
$S^6(1,2)$	83	56	50
$S^6(3,4)$	112	73	50
$S^6(2,3)$	112	73	50
$S^7(4,5)$	219	138	66
$S^7(1\text{-}5)$	72	42	66
$S^7(1,2)$	135	83	66
$S^7(3,4)$	245	150	66
$S^7(2,3)$	219	138	66
$S^8(5,6)$	394	228	78
$S^8(2\text{-}6)$	140	80	78
$S^8(2,3)$	394	228	78
$S^8(4,5)$	520	306	78
$S^8(3,4)$	520	306	78
$S^9(5,6)$	1010	566	118
$S^9(2\text{-}6)$	410	217	118
$S^9(2,3)$	662	381	118
$S^9(4,5)$	1134	656	118
$S^9(3,4)$	1010	566	118
$S^{10}(6,7)$	1832	1009	130
$S^{10}(3\text{-}7)$	840	466	130
$S^{10}(3,4)$	1832	1009	130
$S^{10}(5,6)$	2354	1296	130
$S^{10}(4,5)$	2354	1296	130
$S^{11}(6,7)$	4556	2433	174
$S^{11}(3\text{-}7)$	2147	1108	174
$S^{11}(3,4)$	3137	1675	174
$S^{11}(5,6)$	5082	2740	174
$S^{11}(4,5)$	4556	2433	174
$S^{12}(7,8)$	8318	4330	190
$S^{12}(4\text{-}8)$	4455	2340	190
$S^{12}(4,5)$	8318	4330	190
$S^{12}(6,7)$	10430	5463	190
$S^{12}(5,6)$	10430	5463	190
$S^{13}(7,8)$	20165	10261	218
$S^{13}(4\text{-}8)$	10584	5336	218
$S^{13}(4,5)$	14445	7430	218
$S^{13}(6,7)$	22308	11518	218
$S^{13}(5,6)$	20165	10261	218
$S^{14}(8,9)$	37039	18596	234
$S^{14}(5\text{-}9)$	22022	11262	234
$S^{14}(5,6)$	37039	18596	234
$S^{14}(7,8)$	45476	22877	234
$S^{14}(6,7)$	45476	22877	234
$S^{15}(8,9)$	87947	43198	334
$S^{15}(5\text{-}9)$	50052	24671	286
$S^{15}(5,6)$	65067	32312	286
$S^{15}(7,8)$	96525	47950	286
$S^{15}(6,7)$	87947	43198	286

References

[1] D. L. Dietmeyer, "Generating minimal covers of symmetric function," *IEEE TCAD,* vol. 12, no. 5, pp. 710-713, May 1993.

[2] W. Ke and P. R. Menon, "Delay-testable implementations symmetric functions," *IEEE TCAD*, vol. 14, pp. 772-775, 1995.

[3] S. Chakraborty, S. Das, D. K. Das and B. B. Bhattacharya, "Synthesis of symmetric Functions for path-delay fault testability," *IEEE TCAD*, vol. 19, pp. 1076-1081, September 2000.

[4] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1977.

[5] Y. X. Yang and B. Guo, "Further enumerating Boolean functions of cryptographic significance," *J. Cryptology,* vol. 8, no. 3, pp. 115-122, 1995.

[6] A. Pal and B. B. Bhattacharya, "Syndrome-testable logic design using DSTL arrays for detecting stuck-at and bridging faults," *IEE Proc.,* vol.132, Pt. E, no. 5, 1985.

[7] S. L. Hurst, "Digital summation threshold logic gates: a new circuit element," *IEE Proc.,* vol. 120, no. 11, pp. 1301-1307, 1973.

[8] S. Ghosh and A. K. Choudhury, "Partitions of Boolean functions for realizations with multi-threshold elements," *IEEE Trans. Computers,* vol. C-22, pp. 204-215, 1973.

[9] J. Ja′Ja′ and S.-M. Wu, "A new approach to realize partially symmetric functions," *Tech. Rep. SRC TR 86-54,* Dept. EE, University of Maryland, 1986.