JOURNAL OF ELECTRONIC TESTING: Theory and Applications 22: 125–142, 2006

DOI: 10.1007/s10836-006-6674-3

Implementing Symmetric Functions with Hierarchical Modules for Stuck-at and Path-Delay Fault Testability

HAFIZUR RAHAMAN
Information Technology Dept., Bengal Engg. and Sc. University, Howrah – 711 103, India
rahaman_h@hotmail.com

DEBESH K. DAS
Computer Sci. & Eng. Dept., Jadavpur University, Kolkata –700 032, India
debeshd@hotmail.com

BHARGAB B. BHATTACHARYA
ACM Unit, Indian Statistical Institute, Kolkata-700 108, India
bhargab@isical.ac.in

Received March 2, 2004; Revised December 6, 2005

Editor: K. Chakrabarty

Abstract. A technique for implementing totally symmetric Boolean functions using hierarchical modules is presented. First, a simple cellular module is designed for synthesizing unate symmetric functions. The structure is universal, admits a recursive design, and uses only 2-input AND-OR gates. A universal test set of size $(n^2/8 + 3n/4)$ for detecting all single stuck-at faults can be easily determined for an n-input module, where $n = 2^k$, $k \geq 3$. General symmetric functions are then realized following a unate decomposition method. The synthesis procedure guarantees full robust path-delay fault testability in the circuit. Experimental results on several symmetric functions reveal that the hardware cost of the proposed design is low, and the number of paths in the circuit is reduced significantly compared to those of earlier designs. Results on circuit area and delay for a few benchmark examples are also reported.

Keywords: path-delay fault, stuck-at fault, symmetric Boolean function, synthesis-for-testability, unate function, universal tests

1. Introduction

Synthesis and testing of symmetric Boolean functions received lot of interest in the past [1–4, 7, 9, 11]. They have also applications to reliable data encryption and Internet security [5]. This paper presents a new approach to synthesizing totally symmetric functions using hierarchical modules. We first redesign a well-known cellular module known as *digital summation threshold logic* (DSTL) array [6, 12]. Such an array can be used directly for synthesizing all unate symmetric functions. Further, we show that a test set for detecting all single stuck-at faults in the array can be easily derived directly from its structure without running a test pattern generator. Non-unate symmetric functions can then be synthesized by the method proposed in [3]. A single n-input array augmented by a two-level circuit allows us to design multiple symmetric functions of n-variables.

Failures that cause logic circuits to malfunction at the desired clock rate and violate timing specifications are modeled as delay faults. A circuit is said to have a *path-delay fault*, if the total delay along some path of the circuit exceeds the system clock interval [13, 14]. For each physical path from a primary input to a primary output of the circuit, two logical paths (rising and falling) are usually considered. Detection of a path-delay fault requires a two-pattern test, and this test is said to be *robust*, if it cannot be invalidated by the presence of other path-delay faults in the circuit. A circuit is robustly *delay testable* if and only if *every* detectable path-delay fault has a robust test. It is known that two-level realizations of most of the symmetric functions

are not robustly path-delay testable [2]. Multi-level synthesis techniques in [2] and [3] were targeted to achieve delay fault testability by using 3 to 5 logic levels. The modules based on DSTL array [6] or its modifications [9] are not robustly path-delay testable. In this work, the internal structure of the DSTL array is redesigned such that it can be used to synthesize any general symmetric function with robust path-delay fault testability. The cost of the circuit reduces significantly compared to earlier designs [2, 3]. The number of paths in the circuit also reduces drastically compared to those in [2, 3, 8, 9].

2. Preliminaries

Let $f(x_1, x_2, \cdots, x_n)$ denote a switching function of n Boolean variables. A *vertex (minterm)* is a product of variables in which every variable appears once. The *weight w* of a vertex v is the number of uncomplemented variables that appear in v. A Boolean function is called *negative* (*positive*) *unate*, if each variable appears in *complemented* (*uncomplemented*) form (but not both) in its minimum sum-of-products (s-o-p) expression.

A switching function $f(x_1, x_2, \cdots, x_n)$ is called *totally symmetric* with respect to the variables $(x_1, x_2, \ldots, x_n)$, if it is invariant under any permutation of the variables [4]. Total symmetry can be specified by a set of integers (called a-numbers) $A = (a_i, \ldots, a_j, \ldots, a_k)$, 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 $S^n(a_i, \ldots, a_j, .., a_k)$. A symmetric function is called *consecutive,* if the set A consists of only consecutive integers $(a_l, a_{l+1}, \ldots, a_r)$. 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 $S^n(A) = S^n(A_1) + S^n(A_2) + \cdots + S^n(A_m)$, such that m is minimum and $\forall\, i, j, 1 \le i, j \le m$, $A_i \cap A_j = \emptyset$, whenever $i \neq j$.

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

A function is called *unate symmetric* if it is both unate and symmetric. A unate symmetric function is always consecutive and can be expressed as $S^n(a_l - a_r)$, 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: $S^n(1\text{-}n)$, $S^n(2 - n)$, $S^n(3 - n)$, $\ldots$, $S^n((n-1)-n)$. We express $S^n(n)$ as $u_n(n)$, and $S^n(a_l - a_n)$ as $u_l(n)$ for $1 \le l \le (n-1)$.

Theorem 1 [3]. *A consecutive symmetric function $S^n(a_l - a_r)$, $a_l \neq a_r$, $l < r$, can be expressed as a composition of two unate symmetric functions:*

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

3. Synthesis of Unate Symmetric Functions

Unate symmetric functions can be synthesized by a cellular DSTL array [6]. It is a multi-input and multi-output logic array, consisting of an iterative arrangement of identical cells with a uniform interconnection pattern among them. There are n inputs lines $x_i, 1 \le i \le n$ and n output lines

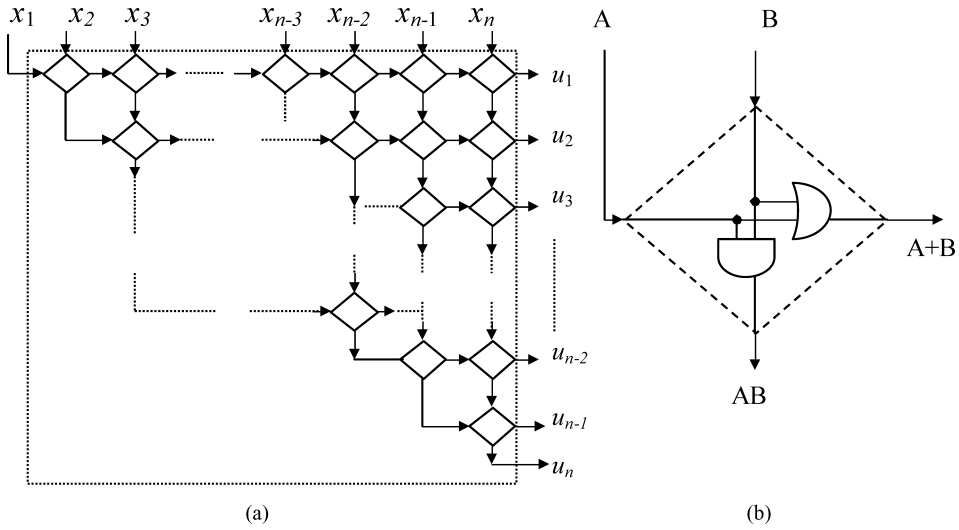

Fig. 1. (a) Basic DSTL array and (b) DSTL cell details.

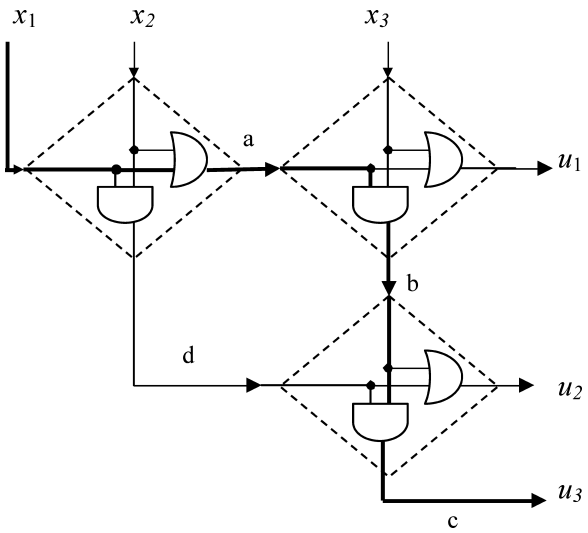

Fig. 2. An example of untestable path-delay fault.

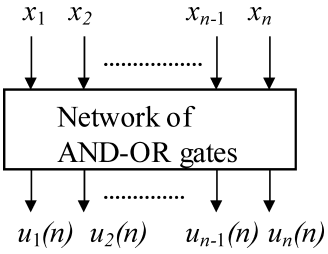

Fig. 3. *Module*(*n*).

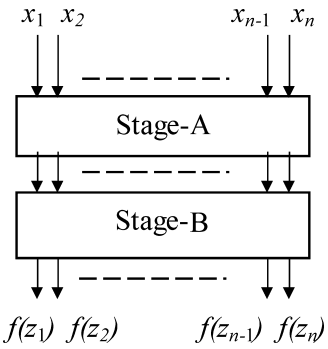

Fig. 4. The structure of *Module*(*n*).

u_j, $1 \le j \le n$. The network is so designed that the output line u_j, $1 \le j \le n$, assumes value 1 if any of the j inputs attain value 1. Hence the output line u_j realizes a threshold function with a threshold value j for the inputs x_i each having an input weight of 1.

The array is shown in Fig. 1(a), where each cell consists of two inputs A and B and produces two outputs A.B (AND) and A+B (OR) as in Fig. 1(b). Each output u_i of the array implements a positive unate symmetric function. The array is a special case of a *balanced inversion parity* (BIP) network [19], i.e., all paths from any primary input to an output

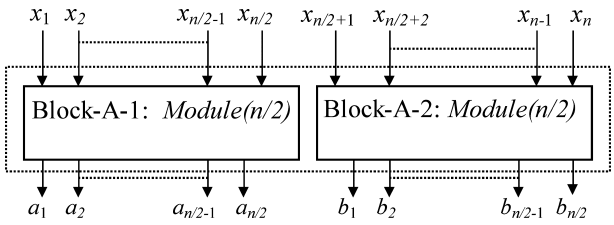

Fig. 5. Stage-A of *Module*(*n*).

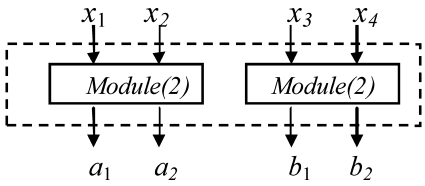

Fig. 6. Stage-A of *Module*(*4*).

have the same inversion parity. It can be easily verified that all multiple stuck-at faults in this array are detectable, and hence the array is irredundant. However, the single-output circuit cone corresponding to an individual output may be redundant. The output functions are as follows (where $\sum$ denotes Boolean OR operation):

$$u_1(n) = S^n(1, 2, 3, \ldots, n) = \sum x_i \quad \text{for } i = 1 \text{ to } n;$$
$$u_2(n) = S^n(2, 3, 4 \ldots, n) = \sum x_i x_j, \quad \text{for } i, j = 1 \text{ to } n;$$
$$u_3(n) = S^n(3, 4, \ldots, n) = \sum x_i x_j x_k, \quad \text{for } i, j, k = 1 \text{ to } n;$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$u_n(n) = S^n(n) = x_1 x_2 \ldots x_{n-1} x_n$$

The following example (Fig. 2) demonstrates that a DSTL array might have an untestable path-delay fault.

Let us consider the path-delay fault (with slow-to-rise or slow-to-fall transition) along the path $x_1 - a - b - c - u_3$ indicated by the bold line. To propagate the transition on primary input x_1, we need x_2 to be 0; to propagate this effect to b, we need x_3 to be 1; and to c, we need d to be 1. However, to make $d = 1$, we must have $x_2 = 1$, which is not possible. Thus, this path-delay fault cannot be detected. Similarly, the path $x_2 - a - b - c - u_3$ is path-delay untestable.

Synthesis of symmetric functions was proposed earlier [9] by redesigning the DSTL array so as to reduce the hardware cost and delay. However, the procedure does not guarantee robust testability of all path-delay faults. All the design procedures reported earlier [6, 8, 9] use a structure called *Module(n)* that has n inputs lines $x_1, x_2, x_3, \ldots, x_n$, and n output functions $u_1(n), u_2(n), u_3(n), \ldots, u_n(n)$ (Fig. 3). Each output u_i, implements a unate symmetric function as described earlier.

Proposed Technique

We first describe a new and simple design of *Module(n)* that has good testability properties.

3.1. Design of Module(n) for n = even

The *Module*(n) is a network of AND-OR gates. For n = even, let $n = 2m$. The *Module(n)* consists of two stages as shown in Fig. 4.

Stage-A: This consists of two blocks – Block A-1 and Block A-2 (Fig. 5), each of which is identical to *Module(n/2)*. The inputs to the Block A-1 (A-2) are $x_1, x_2, x_3, \ldots, x_{n/2}(x_{n/2+1}, x_{n/2+2}, x_{n/2+3}, \ldots, x_n)$. The corresponding output functions are denoted as $a_1, a_2, a_3, \ldots, a_m$ and $b_1, b_2, b_3, \ldots, b_m$, where $m = n/2$.

Clearly, $a_i = \sum x_{j1}x_{j2}\ldots x_{ji}$ for $j1, j2, j3, \ldots, ji = 1$ to $n/2$
and $b_i = \sum x_{j1}x_{j2}\ldots x_{ji}$ for $j1, j2, j3, \ldots, ji = (n/2+1)$ to n.

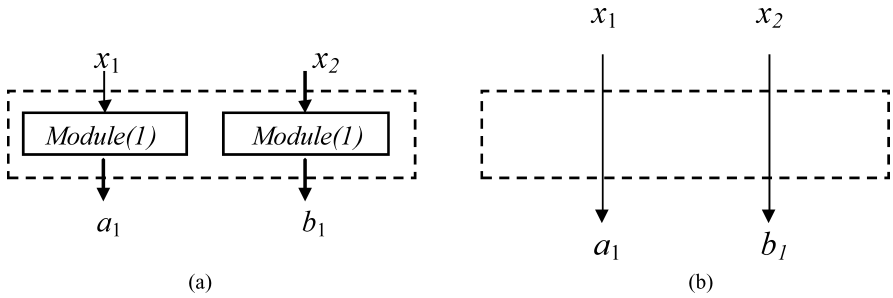

Fig. 7. Stage-A of *Module* (2).

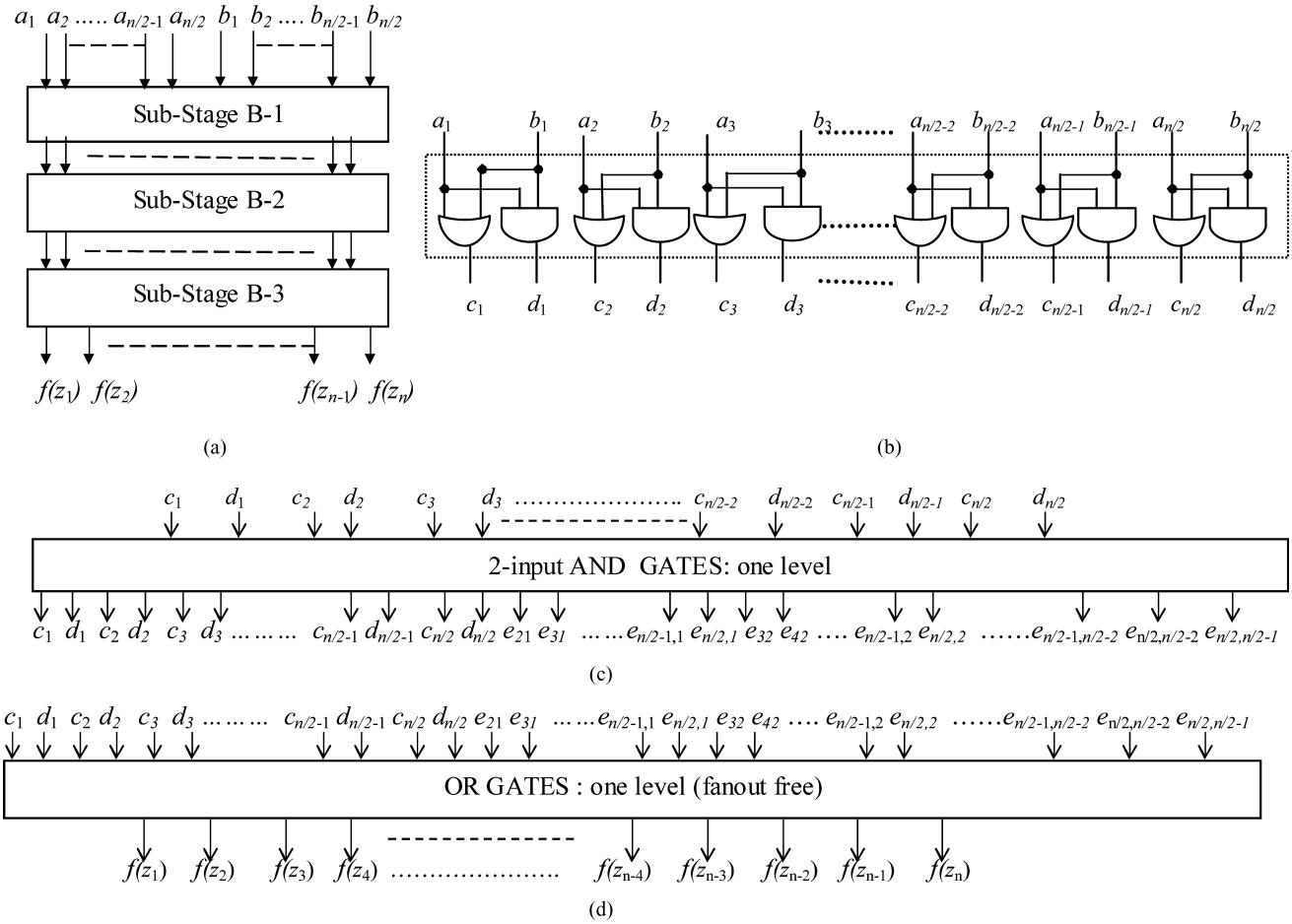

Fig. 8. (a) The structure of Stage-B for n = even, (b) Sub-stage B-1 of *Module* (n), (c) Sub-stage B-2 of *Module* (n) and (d) Sub-stage B-3 of *Module* (n).

Example 2. The stage-A for $n = 4$ is shown in Fig. 6.

Example 3. The stage-A for $n = 2$ is shown in Fig. 7. It contains two copies of *Module*(1), as in Fig. 7(a). But for only one input, there is no gate. *Module*(1) is replaced by a single line, as shown in Fig. 7(b).

Stage-B: The circuit for stage-B, consists of three sub-stages – B-1, B-2 and B-3, as shown in Fig. 8(a). The sub-stage B-1 consists of $n/2$ OR-AND gate pairs (Fig. 8(b)) with n (=2m) inputs ($a_1, a_2, a_3, \ldots, a_m$ and $b_1, b_2, b_3, \ldots, b_m$). The n (=2m) outputs are given by $c_1, d_1, c_2, d_2, c_3, d_3, \ldots, c_{m-1}, d_{m-1}, c_m, d_m$, where and $c_i = a_i + b_i$ and $d_i = a_i b_i$ for $1 \leq i \leq m$. The 1st (nth) output c_1 (d_m) represents $u_1(n)$ (resp. $u_n(n)$).

The sub-stage B-2 consists of only 2-input AND gates of one level. An AND gate having input values c_i and d_j produces the product $c_i d_j$ and all such AND gates with $i > j$ and $1 < i \leq m$ exist in this sub-stage B-2. Its output lines can be classified into two groups—(i) n(=2m) outputs—all the inputs from the outputs of sub-stage B-1 ($c_1, d_1, c_2, d_2, c_3, d_3, \ldots\ldots\ldots, c_{m-1}, d_{m-1}, c_m, d_m$) are passed through B-2, and (ii) $m(m-1)/2$ outputs realizing (c_2d_1), (c_3d_1, c_3d_2), (c_4d_1, c_4d_2, c_4d_3), ……..($c_{m-1}d_1, c_{m-1}d_2, c_{m-1}d_3, \ldots\ldots\ldots c_{m-1}d_{m-3}, c_{m-1}d_{m-2}$), ($c_md_1$, $c_md_2, c_md_3, \ldots\ldots\ldots c_m d_{m-2}, c_m d_{m-1}$). Notice that these outputs may be grouped in $(m-1)$ groups, where the i-th group contains $(m-i)$ elements ($c_{i+1}d_1$, $c_{i+1}d_2, c_{i+1}d_3, \ldots\ldots\ldots\ldots, c_{i+1}d_{i-1}, c_{i+1}d_i$). The scheme is shown in Fig. 8(c), where e_{ij} represents the product $c_i d_j$.

All the outputs from sub-stage B-2 are fed to B-3, which is a one-level fan-out free circuit consisting of $(n-3)$ OR gates. B-3 produces n outputs $f(z_1)$, $f(z_2)$, $f(z_3)$,……, $f(z_{n-2})$, $f(z_{n-1})$, $f(z_n)$, as shown in Fig. 8(d). Three outputs $f(z_1)$, $f(z_{n-1})$ and $f(z_n)$ are obtained by directly passing the signals from the earlier stage, such as $f(z_1) = c_1$, $f(z_{n-1}) = e_{m,m-1} = c_m d_{m-1}$, and $f(z_n) = d_{n/2}$. Except these three, each $f(z_i)$ for $1 < i < n-1$, is realized by an OR-gate, such that the desired functions are produced. The OR gate realizes the function $f(z_i)$ as follows.

$$f(z_i) = c_i + e_{i-1,1} + e_{i-2,2} + e_{i-3,3} + \ldots.. \\ e_{\lfloor (i+1)/2 \rfloor, \lfloor (i-1)/2 \rfloor} + \beta d_i \quad \text{for } i \leq m(= n/2),$$

$$f(z_i) = e_{n/2,i-n/2} + e_{n/2-1,i-n/2+1} + \ldots.... \\ e_{\lfloor (i+1)/2 \rfloor, \lfloor (i-1)/2 \rfloor} + \beta d_i \quad \text{for } i > m(= n/2),$$

where $\beta = 0$ (1) for $i =$ odd (even), $e_{ij} = c_j d_j$.

For $n = 2$, there does not exist any c_i or d_i for $i > 1$. Thus, there is no gate in sub-stage B-2 or B-3. The inputs c_1 and d_1 from sub-stage B-1 are just passed through B-2 and B-3.

Example 4. Figs. 9(a)–9(c) show the sub-stage B-1 for $n = 2$, 4 and 8 respectively.

The algebraic expressions for output functions of the stage-B $f(z_1)$, $f(z_2)$, $f(z_3)$, …, $f(z_i)$, …, $f(z_n)$, in terms of the output functions of the stage-A are given by:

$$f(z_1) = a_1 + b_1$$
$$f(z_2) = (a_2 + b_2) + a_1 b_1$$
$$f(z_3) = (a_3 + b_3) + (a_2 + b_2)\, a_1 b_1$$
$$f(z_4) = (a_4 + b_4) + (a_3 + b_3)\, a_1 b_1 + a_2 b_2$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$f(z_{n-2}) = (a_{n/2} + b_{n/2})\, a_{n/2-2} b_{n/2-2} + a_{n/2-1} b_{n/2-1}$$
$$f(z_{n-1}) = (a_{n/2} + b_{n/2})\, a_{n/2-1} b_{n/2-1}$$
$$f(z_n) = a_{n/2} b_{n/2}$$

More specifically,

$$f(z_i) = a_i + b_i + (a_{i-1} + b_{i-1})\, a_1 b_1 + (a_{i-2} + b_{i-2})\, a_2 b_2 \\ + \cdots\cdots\cdots + (a_{\lfloor (i+1)/2 \rfloor} + b_{\lfloor (i+1)/2 \rfloor})\, a_{(\lfloor (i-1)/2 \rfloor} b_{\lfloor (i-1)/2 \rfloor} \\ + \beta a_{i/2} b_{i/2}$$

where $i \leq n/2$, and $\beta = 0$ (1) if $i =$ odd (even);

$$f(z_i) = (a_{n/2} + b_{n/2})\, a_{i-n/2} b_{i-n/2} \\ + (a_{n/2-1} + b_{n/2-1})\, a_{i-n/2+1} b_{i-n/2+1} + \ldots\ldots\ldots \\ + \left(a_{\lfloor (i+1)/2 \rfloor} + b_{\lfloor (i+1)/2 \rfloor}\right) a_{(\lfloor (i-1)/2 \rfloor} b_{\lfloor (i-1)/2 \rfloor} \\ + \beta a_{i/2} b_{i/2,}$$

where $i > n/2$, and $\beta = 0$ (1) if $i =$ odd (even).

The above expression can be rewritten using a single formula as shown below.

$$f(z_i) = \alpha c_i + \beta d_{i/2} + \sum e_{j,k} \quad \text{for all} \quad i (1 \leq i \leq n)$$

where,

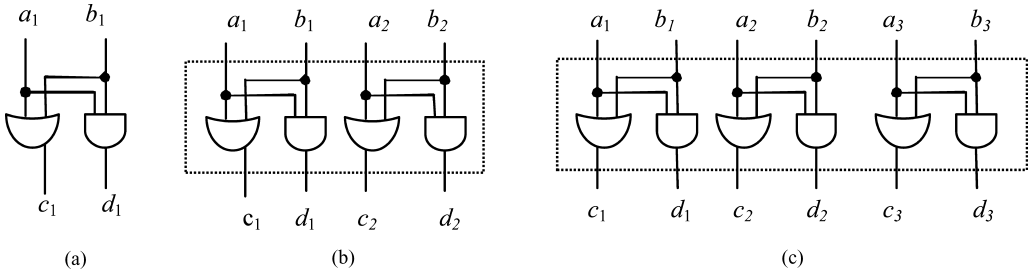

Fig. 9. Sub-stage B-1 of *Module*(n) for (a) $n = 2$ (b) $n = 4$ and (c) $n = 6$.

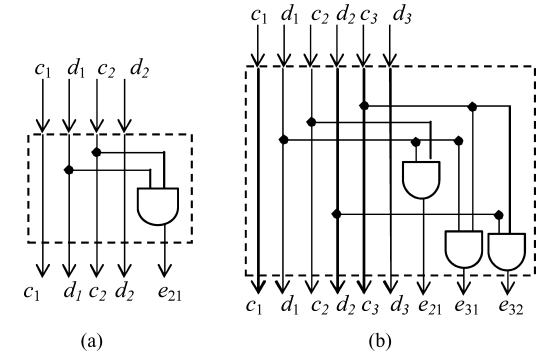

Fig. 10. Sub-stage B-2 of *Module*(*n*) for (a) $n = 4$ (b) $n = 6$.

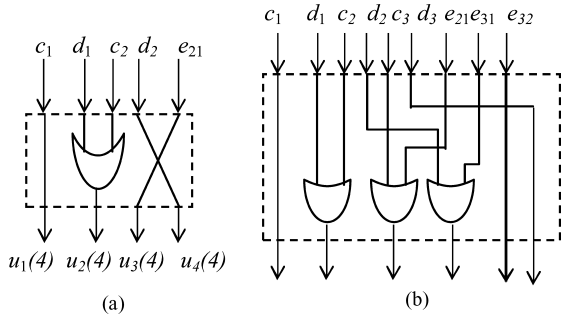

Fig. 11. Sub-stage B-3 of *Module*(*n*) for (a) $n = 4$ (b) $n = 6$.

(a) $\alpha = 0$ (1) for $i > n/2$ ($\leq n/2$)
(b) $\beta = 0$ (1) if $i =$ odd (even).
(c) The range of $\sum$ varies for all j and k satisfying the conditions (i) $j + k = i$ (ii) $k \geq 1$ (iii) $j > k$
(d) $c_l = a_l + b_l$, for any l ($1 \leq l \leq n/2$)
(e) $d_l = a_l b_l$ for any l ($1 \leq l \leq n/2$)
(f) $e_{jk} = c_j d_k$

Thus, d_k ($1 \leq k \leq n/2$) signifies an AND gate realizing $a_k b_k$. Similarly, c_j ($1 \leq j \leq n/2$) signifies an OR gate realizing $a_j + b_j$.

Example 5. The sub-stage B-2 for $n = 4$ ($n = 6$) is shown in Fig. 10(a) and Fig. 10(b).

Example 6. The sub-stage B-3 for $n = 4$ ($n = 6$) is shown in Fig. 11(a) and Fig. 11(b).

Lemma 1. $a_j\, a_k = a_k$ if $k > j$.

Proof: The function $a_j = u_j(n/2) = \sum x_{i1}x_{i2}\ldots x_{ij}$ for $i_1, i_2, i_{3..,}$ $i_j = 1$ to $n/2$. Similarly, $a_k = u_k(n/2) = \sum x_{i1}x_{i2}\ldots.x_{ik}$ for $i_1, i_2, i_3, \ldots, i_k = 1$ to $n/2$. The minimum s-o-p expressions for a_j and a_k are unique and consist of $\binom{n/2}{j}$ and $\binom{n/2}{k}$ product terms respectively. As $k > j$, for every product term P_1 in a_k, there exists a product term P_2 in a_j such that $P_1 \subseteq P_2$. Thus, $a_k \subseteq a_j$. Hence, $a_j\, a_k = a_k$. □

Lemma 2. $b_j b_k = b_k$ *if* $k > j$

Proof: Similar to Lemma 1. □

Lemma 3. $(a_j + b_j)\, a_k b_k = a_j b_k + a_k b_j$ *if* $j > k$

Proof: Follows from Lemmas 1 and 2. □

Lemma 4. *The minimum s-o-p expression for* $\{a_j b_k\}$ *has* $\binom{n/2}{j}$*$\binom{n/2}{k}$ *product terms each with* $(j + k)$ *variables.*

Proof: Clear. □

Lemma 5. *Let* i_1, j_1, i_2, j_2, *be any four variables such that* $1 \leq i_1, j_1, i_2, j_2 \leq n/2$ *and* $i_1 + j_1 = i_2 + j_2$. *Let* P_1 *and* P_2 *be two product terms in the s-o-p expressions of* $a_{i1}b_{j1}$ *and* $a_{i2}b_{j2}$ *respectively. Then* $P_1 \not\subset P_2$ *and* $P_2 \not\subset P_1$.

Theorem 2. *The proposed design of* Fig. 5 *realizes Module(n).*

Proof: From Lemma 3, the function $f(z_i)$ at the output of the 3rd stage is given by:

$$f(z_i) = \sum a_j b_k + a_i + b_i \quad \forall j, k, \text{ such that } j + k = i \text{ and } i \leq n/2$$
$$= \sum a_j b_k \text{ for } \forall j, k, \text{ such that } j + k = i \text{ and } i > n/2.$$

□

Using Lemmas 4 and 5, it follows that $f(z_i)$ is the s-o-p of $\binom{n/2}{i}$ product terms each having i variables. Thus, $f(z_i) = \sum x_{j1}x_{j2}\ldots x_{ji}$ for $1 \leq j_1, j_2, \ldots.., j_i, \leq n = u_i(n)$. Hence the proof.

3.2. Designing Module (n) for n = odd

Let $n = 2m + 1$. *Module(n)* will have two stages, similar to the case when $n =$ even.

Stage-A: This consists of two parts: *Module(m + 1)* and *Module(m)* (Fig. 12(a)). The n (=2m+1) outputs a_1, $a_2, a_3, \ldots\ldots., a_m, a_{m+1}, b_1, b_2, b_3, \ldots\ldots., b_{m-1}, b_m$ of stage-A feed the stage-B.

Stage-B: The circuit for stage-B, can be considered as consisting of three sub-stages – B-1, B-2 and B-3, as shown in Fig. 12(b).

The sub-stage B-1 consists of $n(= 2m+1)$ inputs realizing $(a_1, a_2, a_3, \ldots.., a_m, a_{m+1}$ and $b_1, b_2, b_3, \ldots.., b_m)$. There are $n(= 3m+1)$ outputs, where first $2m$ outputs are given by $c_1, d_1, c_2, d_2, c_3, d_3, \ldots\ldots\ldots, c_{m-1}, d_{m-1}, c_m, d_m$, where $c_i = a_i + b_i$ and $d_i = a_i b_i$ for $1 \leq i \leq m$. The $(m+1)$-th output is obtained by directly passing a_{m+1} through sub-stage B-1. The last m outputs are obtained by directly passing the inputs $(b_1, b_2, b_3, \cdots\cdots, b_m)$ through sub-stage B-1. The scheme is shown in Fig. 12(c).

The sub-stage B-2 consists of only 2-input AND gates of one level. It has n $(=3m+1)$ inputs from sub-stage B-1, as stated above. Its output lines can be classified into two groups—(i) n $(=2m+1)$ outputs—the inputs from the outputs of sub-stage B-1 (c_1, d_1, c_2, d_2, c_3, d_3, ..., $c_{m-1}, d_{m-1}, c_m, d_m$, and a_{m+1}) are passed through B-2, and (ii) $m(m+1)/2$ outputs realizing (c_2d_1), (c_3d_1, c_3d_2), (c_4d_1, c_4d_2, c_4d_3), ...$(c_{m-1}$ d_1, c_{m-1} d_2,c_{m-1} d_3, ... $c_{m-1}d_{m-3}, c_{m-1}d_{m-2})$, $(c_md_1$, c_md_2, c_md_3, ...$c_md_{m-2}, c_md_{m-1})$, $(a_{m+1}b_1$, $a_{m+1}b_2, a_{m+1}b_3$, ...$a_{m+1}b_{m-1}, a_{m+1}b_m)$. Notice that these outputs may be grouped in m groups, where i-th group contains $(m-i)$ elements $(c_{i+1}d_1$, $c_{i+1}d_2, c_{i+1}d_3$, ..., $c_{i+1}d_{i-1}, c_{i+1}d_i)$for i $\leq m$ and $(m+1)$-th group contains $(a_{m+1}b_1$, $a_{m+1}b_2, a_{m+1}b_3$, ..., $a_{m+1}b_{m-1}, a_{m+1}b_m)$. The scheme is shown in Fig. 12(d), where e_{ij} represents the product c_id_j $[a_{m+1}b_j]$ for $i \leq m$ $[= m+1]$.

All the outputs from sub-stage B-2 are fed to B-3, which is a one-level fan-out free circuit consisting of $(n-2)$ OR gates. B-3 produces n outputs $f(z_1)$, $f(z_2)$, $f(z_3)$,....., $f(z_{n-2})$, $f(z_{n-1})$, $f(z_n)$, as shown in Fig. 12(e). Two outputs $f(z_1)$ and $f(z_n)$ are obtained by directly passing through it, such as $f(z_1) = c_1$ and $f(z_n) = a_{m+1}b_m$. Except these two, each $f(z_i)$ for $1 < i < n$, is realized by an OR-gate, to produce the desired functions. The OR gate realizes the function $f(z_i)$ as follows.

$$f(z_i) = c_i + e_{i-1,1} + e_{i-2,2} + e_{i-3,3} + \cdots \cdots e_{\lfloor (i+1)/2 \rfloor, \lfloor (i-1)/2 \rfloor} + \beta d_i \quad \text{for } i \leq m,$$

$$f(z_i) = a_i + e_{i-1,1} + e_{i-2,2} + e_{i-3,3} + \cdots \cdots e_{\lfloor (i+1)/2 \rfloor, \lfloor (i-1)/2 \rfloor} + \beta d_i \quad \text{for } i = m+1,$$

$$f(z_i) = e_{m+1,i-m-1} + e_{m,i-m} + \cdots \cdots .e_{\lfloor (i+1)/2 \rfloor, \lfloor (i-1)/2 \rfloor} + \beta d_i \quad \text{for } i > m(= n/2),$$

where $\beta = 0$ (1) for $i =$ odd (even).

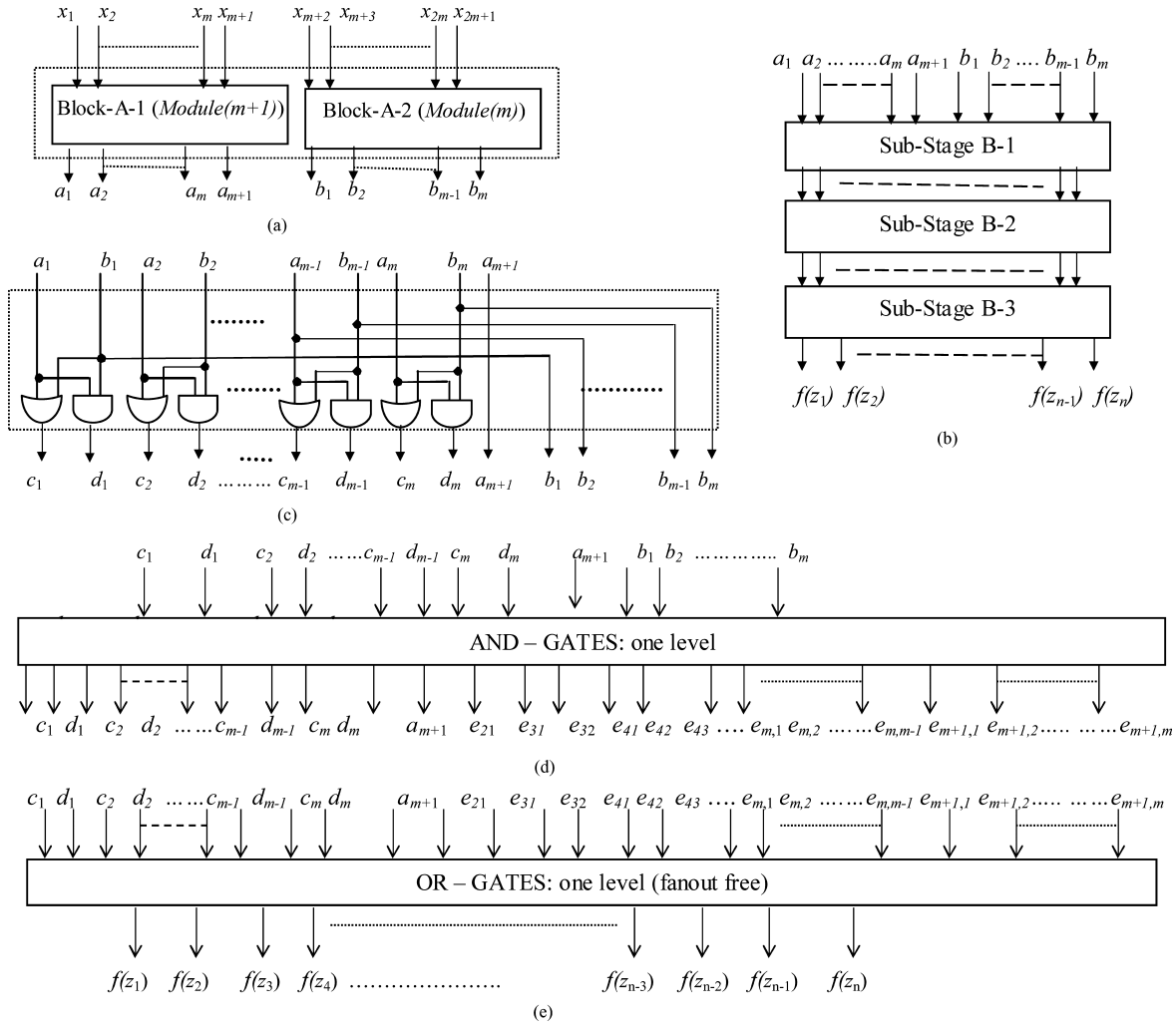

Fig. 12. (a) Stage-A of *Module*(n) for $n = 2m+1$, (b) The structure of Stage-B for odd n, (c) Sub-stage B-1 of *Module*(n) for $n = 2m+1$, (d) Sub-stage B-2 of *Module*(n) for $n = 2m+1$, (e) Sub-stage B-3 of *Module*(n) for $2^{k-1} < \text{n} < 2^k$, with $p = 2^{k-1}$ and $p + m = n$.

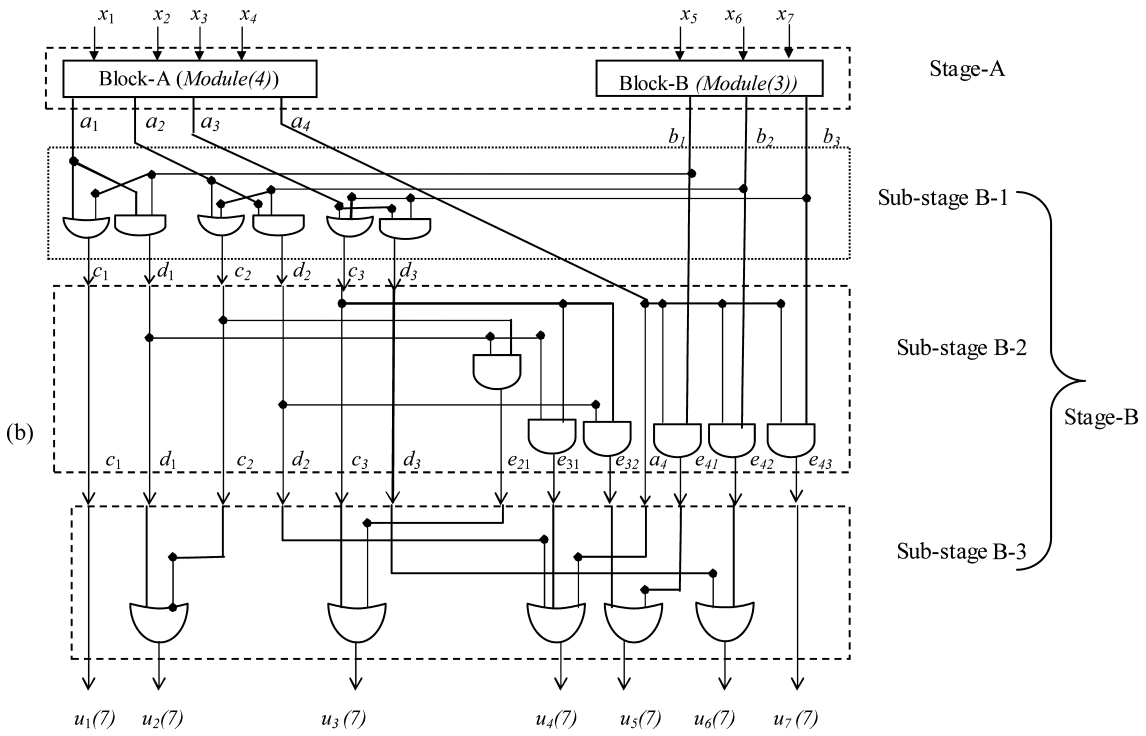

Fig. 13. Module(7).

Thus, for n outputs marked as $f(z_i)$ for $1 \leq i \leq n$, produced by stage-B, each $f(z_i)$ can be represented as

$$f(z_i) = \alpha f_i + \beta d_{i/2} + \sum e_{jk}$$

for all $j$and k satisfying the following conditions

(a) $\alpha = 0$ (1) for $i > n/2$ ($\leq n/2$)
(b) $\beta = 0$ (1) if $i =$ odd (even).
(c) The range of $\sum$ varies for all $j$and k satisfying the conditions (i) $j + k = i$ (ii) $k \geq 1$ (iii) $j > k$
(d) $f_l = a_l + b_l$ [a_l] for any for any l with($1 \leq l \leq m$)[($l = m+1$)]
(e) $d_l = a_l b_l$ for any l($1 \leq l \leq n/2$)
(f) $e_{j,k} = f_j d_k$ [$f_j b_k$] for any j with ($j \leq m$)[($j = m+1$)]

Example 7. Module(7) is shown in Fig. 13.

4. Circuit Cost and Delay

The above design of *Module(n)* reduces the cost of synthesizing a unate symmetric function drastically. The delay as well as cost is smaller than those of the classical DSTL array [6]. The input-to-output path length (delay) can be made equal for each path, if needed, by providing some buffers on the signal paths, without increasing the maximum delay. Such a design can be pipelined using a few latches for fast evaluation of inputs.

4.1. Hardware Cost

Let $C(n)$ denote the number of gates in *Module(n).*

In general, it can be written that $C(n) = C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor) + C_B(n)$, where $C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor)$ represents the number of gates in stage-A and $C_B(n)$ represents the number of gates in the stage-B. For n = even, this leads to $C(n) = 2C(n/2) + C_B(n)$.

Now, $C_B(n) = C_{B1}(n) + C_{B2}(n) + C_{B3}(n)$, where $C_{B1}(n)$, $C_{B2}(n)$ and $C_{B3}(n)$ represent the number of gates in sub-stage $B-1$, $B-2$ and $B-3$ respectively. Let us now calculate these values.

In sub-stage B-1, there are $\lfloor n/2 \rfloor$ pairs of OR-AND gates. Thus $C_{B1}(n) = 2 * \lfloor n/2 \rfloor$.

In sub-stage B-2, there are $\lceil n/2 \rceil * \lceil (n-2)/2 \rceil / 2$ AND gates.

The sub-stage B-3 contains only OR gates. For $n =$ even [odd], this sub-stage contains $(n-3)$ [$(n-2)$] OR gates. Thus the value of $C(n)$ can be summed up.

Except the sub-stage B-3, the remaining part of *Module(n)* contains only two-input gates, where sub-stage B-3 contains OR gates with more than two inputs for any $n \geq 7$. However, for practical reason, the OR gates with large input lines may not be used. To compare our results with those in [6] and [8], let us consider the use of two-input gates only. In this case, only the gate count in sub-stage B-3 will change.

Let the output line of the sub-stage B-3 realizing function $f(z_i)$ be produced by OR-operation of N_i lines from sub-stage B-2. Now, if we consider the use of only two-input

OR gates, to realize $f(z_i)$, the number of OR gates required is, $O_i = N_i - 1$.

Therefore,

$$\begin{aligned} O_i &= \lceil (i-1)/2 \rceil \quad \text{for } i \leq \lfloor n/2 \rfloor \\ &= \lceil (n-i+1)/2 \rceil \quad \text{for } i > \lfloor n/2 \rfloor \text{ and } n = \text{even} \\ &= \lceil (n-i)/2 \rceil \quad \text{for } i > \lfloor n/2 \rfloor \text{ and } n = \text{odd}. \end{aligned}$$

The number of gates in sub-stage B-3 is equal to ΣO_i for $1 \leq i \leq n$.

It can be proved that $\Sigma O_i = \lceil n/2 \rceil * \lceil (n-2)/2 \rceil /2$.

Thus, considering only two-input gates, $C_B(n) = C_{B1}(n) + C_{B2}(n) + C_{B3}(n) = 2 * \lfloor n/2 \rfloor + \lceil n/2 \rceil * \lceil (n-2)/2 \rceil$. It simplifies to $C_B(n) = n(n+2)/4$ $[(n-1)(n+5)/4]$ for n = even [odd].

For $n = 1$, *Module(n)* is a single line. Thus $C(1) = 0$. Using this value for $C(1)$, we can recursively find $C(n)$ for any value of n.

The results on the number of gate counts for $1 \leq n \leq 16$ are shown in Table 2 in comparison to those of [6] and [8].

For $n = 2^k$, $C(n) = 2C(n/2) + n/2\,(n/2+1)$. Considering $C(1) = 0$, recursively, we can get $C(n) = n/2\,(n + \log n - 1)$.

4.2. Circuit Delay

We assume unit gate delay through a 2-input gate. Obviously, the minimum delay $D_{\min}(n)$ is observed at the output $f(z_1)$ and $f(z_n)$, which is given by $D_{\min}(n) = \lceil \log n \rceil$.

Denoting the maximum delay for n as $D_{\max}(n)$, in general we get, $D_{\max}(n), = D_{\max}(\lceil n/2 \rceil) + D^B_{\max}(n)$, where $D^B_{\max}(n)$, denotes the maximum delay in stage-B. Now, $D^B_{\max}(n) = D^{B1}_{\max}(n) + D^{B2}_{\max}(n) + D^{B3}_{\max}(n)$, where $D^{B1}_{\max}(n)$, $D^{B2}_{\max}(n)$ and $D^{B3}_{\max}(n)$ represent the maximum delay in sub-stage *B-1, B-2 and B-3* respectively. From the structural description $D^{B1}_{\max}(n) = 1$ and $D^{B2}_{\max}(n) = 1$. Further, $D^{B3}_{\max}(n) = 1$, if we allow OR gates with more than two inputs in sub-stage B-3.

To find $D^B_{\max}(n)$ using only two-input OR-gates, let an OR gate with more than two inputs are replaced by a tree of two-input OR gates. In that case $D^{B3}_{\max}(n) > 1$ for any $n \geq 7$. It can be observed that $D^B_{\max}(n)$ occurs at the line realizing $f(z_i)$ for $i = \lceil n/2 \rceil$. Now, the output line of the sub-stage B-3 realizing function $f(z_i)$ is produced by OR-operation of N_i lines from sub-stage B-2. The delay contributed by the OR-tree to produce $f(z_i)$ is given as $\lceil log N_i \rceil$, which becomes maximum for $i = \lceil n/2 \rceil$.

From the section 4.1, we can get that $N_{\lceil n/2 \rceil} = \lceil (n+2)/4 \rceil \{\lceil (n+3)/4 \rceil\}$ for n = even $\{$odd$\}$.

Hence the maximum delay $D^B_{\max}(n)$ can be calculated as follows:

$$D_{\max}(n) = D_{\max}(\lceil n/2 \rceil) + \lceil log(N_{\lceil n/2 \rceil}) \rceil,$$

where

$$N_{\lceil n/2 \rceil} = \lceil (n+2)/4 \rceil \{\lceil (n+3)/4 \rceil\} + 2 \quad \text{for } n = \text{even}\{\text{odd}\}.$$

The validity of this formula for $n > 4$ can be justified as follows: for $n = 1$, $D_{\max}(n) = 0$ (Fig. 7(b)); for $n = 2$, there is neither sub-stage B-2 nor B-3, and hence, $D_{\max}(2) = 1$. For $n = 3$, the only AND gate in sub-stage B-2 is produced directly from the inputs of stage-A, thus $D^B_{\max}(3) = 2$, implying $D_{\max}(3) = 3$. For $n = 4$, the only OR gate in sub-stage B-3 is produced directly from the inputs of sub-stage-B-2 (Fig. 11(a)), thus $D^B_{\max}(3) = 2$, implying $D_{\max}(4) = 3$. For $n \geq 5$, all the sub-stages in stage-B will appear. The results on the delay values for $1 \leq n \leq 16$ are shown in Table 2 in comparison to those in [6] and [8].

For $n = 2^k$, $D_{\max}(n) = D_{\max}(n/2) + \lceil log(\lceil (n+2)/4 \rceil) \rceil + 2 = D_{\max}(n/2) + \lceil log(n/4+1) \rceil + 2 = D_{\max}(n/2) + \log n + 1$. Considering $D_{\max}(4) = 3$, recursively, we obtain $D_{\max}(n) = \log n(\log n + 3)/2 - 2$ for $n = 2^k$.

5. Effects on Testability

The proposed design is a balanced inversion parity (BIP) network [19] and implements unate symmetric functions. It admits nice testability properties with respect to both stuck-at and path faults. BIP networks, and in particular, circuits realizing unate functions are known to support universal (i.e. realization independent) test set for stuck-at and delay faults [15–19]. Here, we show that a test set for detecting all single stuck-at faults in the module can be directly derived from its structure without running a test pattern generator. Next, we will show that the module is robustly path-delay testable.

5.1. Testing of Stage-B

Assuming that *Module(n/2)* is testable, let us analyze the testability of the stage-B of *Module(n)*. The problem in observing the test outcome separately in stage-B arises from the fact that we have no direct control to the inputs $a_1, a_2, \ldots . a_{n/2}$, $b_1, b_2, \ldots . b_{n/2}$ to stage-B. Thus, certain combination of logic values may not be available at the inputs of stage-B. If the stage-A is assumed to be fault-free, we can prove the following Lemmas.

Lemma 6. *If for any input pattern* $(x_1, x_2, \cdots \cdots x_{n/2}, x_{n/2+1}, x_{n/2+2}, \ldots ., x_n)$, $a_i[b_i] = 0$, *then* $a_j[b_j] = 0$ *for any* $i < j \leq n/2$.

Lemma 7. *If for any input pattern* $(x_1, x_2, \cdots \cdots x_{n/2}, x_{n/2+1}, x_{n/2+2}, \ldots ., x_n)$, $a_i[b_i] = 1$, *then* $a_j[b_j] = 1$ *for any* $j < i$.

The above two Lemmas are important in the sense that for a test vector of stage-B, its inputs $(a_1, a_2, \ldots..a_{n/2}, b_1, b_2, \cdots\cdots\ b_{n/2})$ cannot violate Lemma 6 and Lemma 7.

5.1.1. Testing of Sub-Stage B-1. Let us consider the testability of the sub-stage B-1, which contains $n/2$ OR gates and $n/2$ AND gates, where the i-th OR (AND) gate has two inputs (a_i, b_i) for $1 \leq i \leq n/2$.

Lemma 8. *All single stuck-at faults at the input and output lines of the OR gates in the sub-stage B-1 are tested by following three combinations*:

$$T_1(n) = \begin{vmatrix} a_1 & a_2 & \cdots\cdots & a_{n/2-1} & a_{n/2} & b_1 & b_2 & \cdots\cdots\cdots & b_{n/2-1} & b_{n/2} \\ 0 & 0 & \cdots\cdots & 0 & 0 & 0 & 0 & \cdots\cdots\cdots & 0 & 0 \\ 0 & 0 & \cdots\cdots & 0 & 0 & 1 & 1 & \cdots\cdots\cdots & 1 & 1 \\ 1 & 1 & \cdots\cdots & 1 & 1 & 0 & 0 & \cdots\cdots\cdots & 0 & 0 \end{vmatrix}$$

Proof: On application of $T_1(n)$, any OR gate (say the i-th OR gate) in the sub-stage B-1 receives all the 3 combinations $(00, 01, 10)$ at its inputs (a_i, b_i) required for testing. Thus, the effect of the fault (error) involving the input or output lines of this OR gate can be propagated to the output line realizing c_i of this sub-stage B-1. The line representing c_i is passed through sub-stage B-2, and is directly connected to one input of an OR gate in sub-stage B-3 producing $f(z_i)$ [for $i = 1$, there is no OR gate, c_1 to be termed as $f(Z_1)$]. The other inputs to this OR-gate of sub-stage B-3 are fed from the AND gates of sub-stage B-2. Now, the output of each AND gate in the sub-stage B-1 is 0 for every input of T_1. For any AND gate in sub-stage B-2, one of its inputs is an output of an AND-gate in sub-stage B-1. Thus, the output of each AND gate in sub-stage B-2 is 0, and except the line c_i, all other lines to the inputs of the OR-gate in sub-stage B-3 realizing $f(z_i)$ are at 0. So, the error on c_i can be propagated to $f(z_i)$. Hence the proof. □

Lemma 9. *Any single stuck-at fault at the input and output lines of the p-th AND gate in sub-stage B-1 is tested by the following three combinations:*

$$t_2^{p-AND} = \begin{vmatrix} a_1 & a_2 & \cdots\cdots & a_{p-1} & a_p & a_{p+1} & \cdots\cdots & a_{n/2-1} & a_{n/2} & b_1 & b_2 & \cdots\cdots & b_{p-1} & b_p & b_{p+1} & \cdots\cdots\cdots & b_{n/2-1} & b_{n/2} \\ 0 & 0 & \cdots\cdots & 0 & 0 & 0 & \cdots\cdots & 0 & 0 & 1 & 1 & \cdots\cdots & 1 & 1 & 1 & \cdots\cdots\cdots & 1 & 1 \\ 1 & 1 & \cdots\cdots & 1 & 1 & 1 & \cdots\cdots & 1 & 1 & 0 & 0 & \cdots\cdots & 0 & 0 & 0 & \cdots\cdots\cdots & 0 & 0 \\ 1 & 1 & \cdots\cdots & 1 & 1 & 0 & \cdots\cdots & 0 & 0 & 1 & 1 & \cdots\cdots & 1 & 1 & 0 & \cdots\cdots\cdots & 0 & 0 \end{vmatrix}$$

Proof: On application of t_2^{p-AND}, the p-th AND gate in the sub-stage B-1 receives all the 3 combinations (01, 10, 11) at its inputs (a_p, b_p) required for its testing. Thus, an error on an input or an output line of this AND gate can be propagated to the output line realizing d_p of the sub-stage B-1. The question is whether this error can be propagated to the output. For $p = n/2$, the line d_p is directly connected to the output, thus the error is propagated. Let us consider the case for $p < n/2$. □

For (01 and 10) tests: The line representing d_p fans out in sub-stage B-2 and one of its branch lines is AND-ed with the line realizing $c_{n/2} = a_{n/2} + b_{n/2}$. The output line $e_{n/2.p}$ of this AND-gate realizing $c_{n/2}d_p$ is fed to an OR-gate in sub-stage B-3 realizing the function $f(z_{n/2+p})$. The other inputs of this OR gate realize either $c_j d_k = (a_j + b_j)a_k b_k$ with $j + k = n/2 + p$, or d_k [for $2k = n/2 + p$]. The line d_k realizes the function $a_k d_k$. Clearly, $d_k = 0$ for the fist two inputs of t_2^{p-AND}. Thus, except the line $e_{n/2+p}$, all other lines to the inputs of the OR-gate in sub-stage B-3 realizing $f(z_{n/2+p})$ are at logic 0. So, the error on $e_{n/2+p}$ can be propagated to $f(z_{n+2+p})$.

For (11) test: The line representing d_p fans out in sub-stage B-2 and one of its branch lines is directly connected to the OR-gate of sub-stage B-3 realizing $f(z_{2p})$. As stated earlier, the i-th output of *Module(n)* $f(z_i)$ for $i = 2p$ can be expressed as follows: $f(z_i) = [c_{2p}+] \sum c_j d_k + d_p$, for $[i \leq n/2]$ $i > n/2$, where j and k satisfy the following conditions – (i) $k > 0$ (ii) $d_k = a_k b_k$ (iii) $j + k = 2p$ (iv) $j > k$ (v) $c_j = a_j + b_j$. Now, condition (iii) and (iv) imply that $j > p$. For the third input in t_2^{p-AND}, $a_j = 0$, $b_j = 0$ for $j > p$. It implies that except the line realizing d_p, all other lines to the inputs of the OR-gate in sub-stage B-3 realizing $f(z_{2p})$ are at logic 0. So the error on d_p can be propagated to $f(z_{2p})$.

5.1.2. Testing of Sub-Stage B-2. The stage B-2 contains $n/4(n/2 - 1)$ two-input AND gates. Consider, an AND gate with two inputs c_j and d_k, producing $c_j d_k$. Notice that $j > k$ and c_j and d_k are respectively the output function of an OR and an AND gate in sub-stage B-1, producing $a_j + b_j$ and $a_k b_k$. Moreover, as $j > k$, j must be at least 2.

Lemma 10. *The input and output lines of the AND gate producing $c_j d_k$ in sub-stage B-2 are tested by following three combinations (for $j > 1$ and also $j > k$).*

$(a_1 a_2 \ldots\ldots a_{k-1} a_k a_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots a_{n/2-1} a_{n/2})$
$(b_1 b_2 \ldots\ldots b_{k-1} b_k b_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots b_{n/2-1} b_{n/2})$
(i) 01 test: (1 1 1 1 0……..0 0 0……0 0)
(1 1……1 1 0……. 0 0 0……0 0)
(ii) 10 test: this can be done by any of the following two tests:
$(a_1 a_2 \ldots\ldots a_{k-1} a_k a_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots a_{n/2-1} a_{n/2})$
$(b_1 b_2 \ldots\ldots b_{k-1} b_k b_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots b_{n/2-1} b_{n/2})$
(1 1 1 1 1……..1 1 0……0 0) (0 0……0 0 0…….0 0 0……0 0)
or, (0 0 0 0 0……..0 0 0……0 0) (1 1……1 1 1……. 1 1 0……0 0)
11 test: this can be done by any of the following two tests.
$(a_1 a_2 \ldots\ldots a_{k-1} a_k a_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots a_{n/2-1} a_{n/2})$
$(b_1 b_2 \ldots\ldots b_{k-1} b_k b_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots b_{n/2-1} b_{n/2})$
(1 1 1 1 1……..1 1 0……0 0) (1 1……1 1 0……. 0 0 0……0 0)
or, (1 1 1 1 0……..0 0 0……0 0) (1 1……1 1 1……. 1 1 0……0 0)

Proof: It can be easily observed that the AND gate with input lines realizing c_j and d_k receives the required test inputs (01, 10, 11) as mentioned above. Thus, the error due to any stuck-at fault at the input or output lines of the AND gate is propagated to the output line e_{jk} realizing the function $c_j d_k$. This output line is fed to an input of an OR gate, which realizes the function $f(z_i)$ where $i = j + k$. □

The value $f(z_i)$is given by

$$f(z_i) = c_i + \sum c_m d_n [+d_{i/2}] \quad \text{for } i \leq n/2 \quad \text{and } i = \text{odd [even]}$$
$$= \sum c_m d_n [+d_{i/2}] \quad \text{for } i > n/2 \quad \text{and } i = \text{odd [even]},$$

and for all possible combinations of m and n, such that $n > 0$, $m > n$ and $m + n = i = j + k$.

Without loss of generality, let us consider the case with $f(z_i) = c_i + \sum c_m d_n + d_{i/2}$. It can be expressed as $f(z_i) = c_i + c_{i-1}d_1 + c_{i-2}d_2 + \ldots\ldots + c_{j+1}d_{k-1} + c_j d_k + c_{j-1}d_{k+1} + \ldots\ldots + c_{i/2+1}d_{i/2-1} + d_{i/2}$. In the sub-stage B-3, one input to the OR gate realizing $f(z_i)$ represents the function $c_j d_k$, while the other inputs realize $c_i, c_{i-1}d_1, c_{i-2}d_2, \ldots\ldots\ldots c_{j+2}d_{k-2}, c_{j+1}d_{k-1}, c_{j-1}d_{k+1}, \ldots\ldots\ldots, c_{i/2+1}d_{i/2-1}, d_{i/2}$, where for any $p, c_p = a_p + b_p$ and $d_p = a_p b_p$. It can be observed that for all the combinations of $(a_1 a_2 \ldots\ldots a_{k-1} a_k a_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots a_{n/2-1} a_{n/2})$ $(b_1 b_2 \ldots\ldots b_{k-1} b_k b_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots b_{n/2-1} b_{n/2})$, each of $c_i, c_{i-1}d_1, c_{i-2}d_2, \ldots\ldots\ldots c_{j+2}d_{k-2}, c_{j+1}\ d_{k-1}, c_{j-1}d_{k+1}, \ldots\ldots\ldots, c_{i/2+1}d_{i/2-1}, d_{i/2}$, is at logic 0. Thus the error on the line e_{jk} realizing $c_j d_k$ can be propagated to the output of the OR gate realizing $f(z_i)$.

Lemma 11. *To apply (11) test to the AND gate in sub-stage B-2 with inputs $c_{n/2}$ and $d_{n/2-1}$, the following combinations can be used:*

$(a_1 a_2 \ldots\ldots a_{k-1} a_k a_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots a_{n/2-1} a_{n/2})$
$(b_1 b_2 \ldots\ldots b_{k-1} b_k b_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots b_{n/2-1} b_{n/2})$
(1 1 1 1 1……..1 1 1……1 1) (1 1……1 1 1……. 1 1 1……1 1)

Proof: The concerned AND gate receives the input (11) in the above pattern. Now, the output of this AND gate is not passed through any OR gate in sub-stage B-3 as it is directly connected to the output line $f(z_{n-1})$. Thus, the above pattern is able to detect the faults corresponding to (11) test. □

5.1.3. Testing of Sub-Stage B-3

Lemma 12. *If a test set T is sufficient to detect the faults in sub-stages B-1 and B-2, then T is also sufficient to detect any fault in sub-stage B-3.*

Proof: The sub-stage B-3 is a one-level circuit with OR gates only. Now, as T detects the faults at the output lines of sub-stage B-1 and B-2, it implies that faults on all input lines to the OR gates of sub-stage B-3 are detected by the test set T. As there is no fan-out in the sub-stage B-3, T contains the set of vectors that detect the faults in sub-stage B-3. □

5.1.4. Complete Test Set of Stage-B.

From Lemmas 8–11, it follows that a pattern $(a_1 a_2 \ldots\ldots a_{k-1} a_k a_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots a_{n/2-1} a_{n/2})$ $(b_1 b_2 \ldots\ldots b_{k-1} b_k b_{k+1} \ldots\ldots a_{j-1} a_j a_{j+1} \ldots\ldots b_{n/2-1} b_{n/2})$, which is required for detecting a fault, is a valid combination in accordance to Lemma-6 and Lemma-7. Thus, these values of a_i's and b_i's can be obtained by controlling the input lines $(x_1 x_2 x_3 \ldots\ldots\ldots x_{n/2-2} x_{n/2-1} x_{n/2})$, and $(x_{n/2+1} x_{n/2+2} x_{n/2+3} \ldots\ldots\ldots x_{n-2} x_{n-1} x_n)$ respectively. For example, if we like to obtain $a_i = 1$ for all $i < p$ and $a_i = 0$ for all $i > p$, then among the $n/2$ input lines $(x_1 x_2 x_3 \ldots\ldots\ldots x_{n/2-2} x_{n/2-1} x_{n/2})$, we have to choose any p lines to be set to logic 1, and the other $(n/2 - p)$ lines to be set to 0. A similar case happens for b_i's, where we have to set the input lines $(x_{n/2+1} x_{n/2+2} x_{n/2+3} \ldots\ldots\ldots x_{n-2} x_{n-1} x_n)$.

Definition 1. An input vector I = [x_1, x_2, x_3,…, $x_{n/2-2}$, $x_{n/2-1}$, $x_{n/2}$, $x_{n/2+1}$, $x_{n/2+2}$, $x_{n/2+3}$,……x_{n-2},x_{n-1},x_n] for $n = 2^k$, is said to have a weight-pair denoted by ($w_1(n)$, $w_2(n)$) where $w_1(n)[w_2(n)]$ is an integer value representing the number of 1s among (x_1, x_2, x_3,……$x_{n/2-2}$, $x_{n/2-1}$) [($x_{n/2+1}$, $x_{n/2+2}$, $x_{n/2+3}$,……x_{n-2}, x_{n-1}, x_n)]. The weight-sum of the vector I denoted by $w_{\text{sum}}(n)$ is defined as the summation ($w_1(n) + w_2(n)$). The smaller of ($w_1(n)$, $w_2(n)$) is denoted as $w_{\min}(n)$.

The following corollaries are apparent from Lemmas-8, 9, 10 and 11 respectively.

Corollary 1. *All stuck-at faults at the input and output lines of the OR gates in the sub-stage B-1 are tested by the three input vectors having weight-pairs* $(w_1(n), w_2(n))$: *(i)* $(0, 0)$, *(ii)* $(0, n/2)$ *and (iii)* $(n/2, 0)$ *with weight-sums* $w_{\text{sum}}(n)$: *(i)* 0 *(ii)* $n/2$ *and (iii)* $n/2$ *respectively.*

Corollary 2. *Any single stuck-at fault at the input and output lines of the* i*-th AND gate in sub-stage B-1 is tested by the three input vectors having weight-pairs* $(w_1(n), w_2(n))$: *(i) (0, n/2), (ii)* $(n/2, 0)$ *and (iii)* (i, i) *with weight-sums* $w_{\text{sum}}(n)$: *(i)* $n/2$ *(ii)* $n/2$ *and (iii)* $2i$ *respectively.*

Corollary 3. *The input and output lines of the AND gate producing* $c_j d_k$ *(for* $j > 1$ *and also* $j > k$*) in sub-stage B-2 are tested by the three input vectors having weight-pairs* $(w_1(n), w_2(n))$: *(i)* (k, k), *(ii)* $(j, 0)$ *or* $(0, j)$ *and (iii)* (j, k)*or* (k, j) *with weight-sums* $w_{\text{sum}}(n)$: *(i)* $2k$ *(ii)* $2j$*and (iii)* $(j + k)$ *respectively.*

Corollary 4. *To apply the (11) test to the AND gate in sub-stage B-2 with inputs* $c_{n/2}$ *and* $d_{n/2-1}$*, the test vector I having weight-pairs* $(w_1(n), w_2(n))$ *and weight-sum* $w_{\text{sum}}(n) = n$ *can be used.*

Thus, Lemma 11 and Corollary 4 imply that the input vector with $(w_1(n), w_2(n))$*: (n/2*$-1, n/2$*)* or *(n/2,n/2*-1*)* as described by Corollary 3 is not required for testing the AND gate with inputs $c_{n/2}$ and $d_{n/2-1}$.

Instead of expressing a test pattern t as an $n-$bit vector $t^{\text{Bool}}(n)$, let us express it as a 4-tuple $t^w(n)$*:* $\{w_{min}(n), w_1(n), w_2(n), w_{\text{sum}}(n)\}$.Clearly, for a given $t^w(n)$, there may exist several vectors $t^{\text{Bool}}(n)$. Corollaries 1 to 4 state that the test vectors with weight functions $w_1(n)$ and $w_2(n)$ will be sufficient to detect a fault in stage-B. A test vector $t^w(n)$ with weight values $(w_1(n), w_2(n))$, implies that we can choose $t^{\text{Bool}}(n)$ as any combination of values in$(x_1, x_2, x_3, \ldots\ldots, x_{n-2}, x_{n-1}, x_n)$such that among the first $n/2$ input lines $(x_1, x_2, x_3, \ldots\ldots, x_{n/2-2}, x_{n/2-1}, x_{n/2})$, any $w_1(n)$ lines can be set to 1 and others to 0, and among the last $n/2$ input lines $(x_{n/2+1}, x_{n/2+2}, x_{n/2+3}, \ldots\ldots\ldots, x_{n-2}, x_{n-1}, x_n)$, any $w_2(n)$ lines can be set to 1 and others to 0. This provides us more flexibility in choosing the test vectors.

Lemma 13. *The complete test set to detect the faults at stage-B is given by the set* T_B^w *of input vectors with their* $w_{min}(n)$ *weight pairs* $(w_1(n), w_2(n))$*, and weight-sums* $w_{\text{sum}}(n)$ *as shown:*

$$T_B^w(n) = \left|\begin{array}{cccc} w_{min}(n) & w_1(n) & w_2(n) & w_{\text{sum}}(n) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 3 & 0 & 3 \\ 0 & 0 & 4 & 4 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & n/2-1 & 0 & n/2-1 \\ 0 & 0 & n/2 & n/2 \\ 0 & n/2 & 0 & n/2 \\ 1 & 1 & 1 & 2 \\ 1 & 2 & 1 & 3 \\ 1 & 1 & 3 & 4 \\ 1 & 4 & 1 & 5 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & n/2-1 & n/2 \\ 1 & n/2 & 1 & n/2+1 \\ 2 & 2 & 2 & 4 \\ 2 & 3 & 2 & 5 \\ 2 & 2 & 4 & 6 \\ 2 & 5 & 2 & 7 \\ \vdots & \vdots & \vdots & \vdots \\ 2 & n/2-1 & 2 & n/2+1 \\ 2 & 2 & n/2 & n/2+2 \\ \vdots & \vdots & \vdots & \vdots \\ n/2-3 & n/2-3 & n/2-3 & n-6 \\ n/2-3 & n/2-2 & n/2-3 & n-5 \\ n/2-3 & n/2-3 & n/2-1 & n-4 \\ n/2-3 & n/2 & n/2-3 & n-3 \\ n/2-2 & n/2-2 & n/2-2 & n-4 \\ n/2-2 & n/2-2 & n/2-1 & n-3 \\ n/2-2 & n/2 & n/2-2 & n-2 \\ n/2-1 & n/2-1 & n/2-1 & n-2 \\ n/2 & n/2 & n/2 & n \end{array}\right|$$

Proof: It is easy to observe that the above set contains all the input combinations necessary for testing of stage-B, as described in Corollaries 1, 2, 3 and 4. □

Example 8. $T_B^w(2)$ and $T_B^w(4)$ are given as follows.

$$T_B^w(2) = \left|\begin{array}{cccc} w_{min}(2) & w_1(2) & w_2(2) & w_{\text{sum}}(2) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 2 \end{array}\right|$$

$$T_B^w(4) = \left|\begin{array}{cccc} w_{\min}(4) & w_1(4) & w_2(4) & w_{\sum}(4) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 4 \end{array}\right|$$

Example 9. $T_B^{\text{Bool}}(4)$ can be derived from $T_B^w(4)$. Notice that for $(w_1(4), w_2(4)) = (1, 1)$, there are four possible Boolean vectors $(x_1x_2x_3x_4) = (0\ 1\ 0\ 1)$ or $(0\ 1\ 1\ 0)$ or $(1\ 0\ 0\ 1)$ or $(1\ 0\ 1\ 0)$. Thus, $T_B^{\text{Bool}}(4)$ contains 5 vectors, given by (0000, 0011, 1100, 0101 or 0110 or 1001 or 1010, 1111).

Example 10. $T_B^{\text{Bool}}(2)$ can be derived from $T_B^w(2)$ as (00, 01, 10,11).

Notice that for each vector in $T_B^w(n)$, the pair $(w_1(n), w_2(n))$is distinct and thus there is no chance of repetition of any vector. However, some other test set for detecting the faults at stage-B with the same or larger size (but not of smaller size) can be derived from Corollaries 1, 2, 3 and 4. However, the particular set $T_B^w(n)$ chosen above has some interesting properties as described below.

Characteristics-1: If $w_{\min}(n) = i$, there will be a vector with $(w_1(n), w_2(n)) = (i + 1, i)$, for all i, except in two cases: for $i = 0$ and $i = n/2$. The first exception occurs as the Corollary 3 is valid for $j > 1$. The second exception occurs because the test obtained by applying Corollary-4 obviates the necessity of the vector with $(w_1(n), w_2(n)) = (n/2,n/2\text{-}1)$ as stated in Corollary 3.

Characteristics-2: For a given value of $w_{\min}(n)$, the values of $w_{\sum}(n)$'s may be different, except in the case for $w_{\min}(n) = 0$ and $w_{\sum}(n) = n/2$, as it appears twice with$(w_1(n), w_2(n)) = (0,n/2)$ and $(n/2,0)$. It appears as both the vectors with$(w_1(n), w_2(n)) = (0,n/2)$ and$(w_1(n), w_2(n)) = (n/2,0)$ are required for detection of faults in OR gates of sub-stage B-1 (Corollary 1).

Characteristics-3: For a vector, if $w_{\sum}(n) =$ even, then $w_1(n) = w_{\min}(n)$, otherwise $w_2(n) = w_{\min}(n)$, except in only one case, where $(w_1(n), w_2\ (n)) = (n/2,0)$.

Characteristics-4: In $T_B^w(n)$, $w_{\sum}(n) = i$ with $0 \le i \le n/2$ occurs $\lceil (i + 1)/2 \rceil [\lceil (n - i+1)/2 \rceil]$ times for $i \le n/2 [> n/2]$, only with two exceptions, when $w_{\sum}(n) = n - 1$ and $w_{\sum}(n) = 1$.

Characteristics-5: Except in two cases, for any value of i with $0 \le i \le n/2$, $w_1(n)$ can become equal to $i(n/4+1)$ times in $T_B^w(n)$; a similar property is also true for $w_2(n)$. The exceptional cases include: the occurrence of $w_1(n) =1$ and that of $w_2(n) = n - 1$, each $n/4$ times.

The above discussions lead to following Lemma.

Lemma 13. *For $n = 2^k$, $k > 2$, the size of $T_B^w(n)$ is $(n^2/8+ 3n/4)$.*

Remark: For $k =1$, $n = 2$, and$T_B^w(2) = 4$, as already shown in Example-8.

5.2. Testing of Stage-A

The stage-A of *Module(n)* contains two copies of *Module(n/2),* where the outputs of the circuit on the left [right] feed the inputs $(a_1\, a_2 \ldots\ldots \ldots a_{j-1}a_j a_{j+1} \ldots\ldots a_{n/2-1}a_{n/2})$ $[(b_1 b_2 \ldots\ldots a_{j-1}a_j a_{j+1} \ldots\ldots b_{n/2-1}b_{n/2})]$ of stage-B of *Module(n)*. The input values $(x_1x_2x_3 \ldots\ldots\ldots\ldots x_{n/2-2}$ $x_{n/2-1}x_{n/2})$ $[(x_{n/2+1}x_{n/2+2}x_{n/2+3} \ldots\ldots\ldots x_{n-2}x_{n-1}x_n)]$ determining $w_1(n)$ $[w_2(n)]$ produce these inputs $(a_1a_2 \ldots\ldots\ldots a_{j-1}$ $a_j a_{j+1} \ldots\ldots a_{n/2-1}a_{n/2})$ $[(b_1$ $b_2 \ldots\ldots b_{k-1}b_k b_{k+1} \ldots\ldots a_{j-1}$ $a_j a_{j+1} \ldots\ldots b_{n/2-1}$ $b_{n/2})]$ to stage-B. Assume that a test vector t $(x_1, x_2,$ $x_3 \ldots\ldots\ldots, x_{n-2}, x_{n-1}x_n)$ with weight-pair $(w_1(n),$ $w_2(n))$ detects a fault in stage-B of *Module(n).* The question is, whether we can find a pattern in t such that in $(x_1x_2x_3 \ldots\ldots\ldots\ldots x_{n/2-2}x_{n/2-1}x_{n/2})$ $[(x_{n/2+1}$ $x_{n/2+2}$ $x_{n/2+3} \ldots\ldots\ldots x_{n-2}x_{n-1}x_n)]$ there are 1s at exactly $w_1(n)\,[w_2(n))]$ positions. The following Lemma answers this question.

Lemma 14. *For $n = 2^k$, using proper distribution of 0s and 1s, $T_B^w(n)$ can be made to include $T_B^w(n/2)$, when $k \ge 3$. Thus, the test set detecting the faults in stage-B of Module(n) can be so chosen that it will detect all the single stuck-at faults in stage-B of Module(n/2).*

Proof: If an n-bit input vector t with weight-pair $(w_1(n), w_2(n))$ is applied to *Module(n),* it means that a vector with $n/2$ inputs is applied to the both the *Module(n/2)* in stage-A with the left [right] Module having the weight-sum $w_{\sum}(n/2) = w_1(n)\ [w_2(n)]$. □

Now, we can easily obtain the set $T_B^w(n/2)$ from Lemma 12. Following the characteristics of $T_B^w(n)$, we can say that in $T_B^w(n/2)$, $w_{\sum}(n/2)$ appears as 0 or n for one time, $w_{\sum}(n/2)$ appears as 2 or $n - 2$ for 2 times, $w_{\sum}(n/2)$ as 3 or $n - 3$ for 3 times, and so on. In this manner, $w_{\sum}(n/2)$ appears as $n/4$, $(n/8 + 1)$ times (maximum).

For any value of i with $0 \le i \le n/2$ the incidence of $w_1(n) = i$ or $w_2(n) = i$ occurs at least $n/4$ times. As $n/4 > n/8+1$ for $n \ge 8$, for every required value of $w_{\sum}(n/2)$ in $T_B^w(n/2)$ can be obtained from $w_1(n)$ or $w_2(n)$. Hence the proof.

5.3. The Complete Test Set T(n) for Module(n)

We consider the two representations of $T(n)$: $T^w(n)$ and $T^{\text{Bool}}(n)$, expressing the set with weights and Boolean values respectively. Let $T_A(n)$ be the test set for detecting the faults in stage-A, which has two representations $T_A^w(n)$

and $T_A^{\text{Bool}}(n)$. Obviously, $T^w(n)[T^{\text{Bool}}(n)]$ must include both $T_B^w(n)$ $[T_A^{\text{Bool}}(n)]$ and $T_A^w(n)$ $[T_A^{\text{Bool}}(n)]$. Now, stage-A contains two circuits realizing *Module(n/2).* Each such circuit realizing *Module(n/2)* requires a test set that includes $T_B^w(n/2)$ and $T_A^w(n/2)$. It has been stated in Lemma 13 that for $n = 2^k$, $k \geq 3$, $T_B^w(n)$ can be chosen to include $T_B^w(n/2)$ for the two modules *Module(n/2)* in Stage-A of *Module(n).* Similarly, the set $T_B^w(n/2)$ in each *Module(n/2)* can be chosen to include $T_B^w(n/4)$ of the stage-A of all the modules realizing *Module(n/2)*. Iterating this process, we can conclude that for any $k \geq 3$, $T_B^w(2^k)$ of *Module*(2^k) can be so chosen that for any sub-circuit in *Module* (2^k) realizing *Module*(2^i), $T_B^w(2^i)$ can be constructed from $T_B^w(2^k)$ for any i with $2 \leq i < k$.

Lemma 15. $T^{\text{Bool}}(2) = T_B^{\text{Bool}}(2)$.

Proof: The stage-A of *Module(2),* does not contain any gate, but only two signal lines (Fig. 7(b)). The test for any single line requires two test vectors 0 and 1 on that line. The set $T_B^{\text{Bool}}(2)$ delivers such bits to the two input lines. Thus no additional test is required for stage-A of *Module(2),* so $T^{\text{Bool}}(2) = T_B^{\text{Bool}}(2)$. □

Lemma 16. *For Module(8),* $T_B^{\text{Bool}}(8)$ *can always be chosen such that* $T^{\text{Bool}}(8) = T_B^{\text{Bool}}(8)$.

Proof: Following Lemma 12, $T_B^w(8)$ contains 14 vectors with weight functions $(w_1(8), w_2(8))$:(0,0), (0,2), (3,0), (0,4), (4,0), (1,1), (2,1), (1,3), (4,1), (2,2), (3,2), (2,4), (3,3) and (4,4). We now have to set 0s and 1s to the input variables. In the stage-A of *Module(8),* there are two copies of *Module(4).* For any test vector $t^w(8)$ with weight pair $(w_1(8), w_2(8))$, $w_1(8)[w_2(8)]$ becomes equal to $w_{\text{sum}}(4)$ of the left [right] *Module*(4) of stage-A. Now among the 14 vectors of $T_B^w(8)$, let us consider the distribution of a subset $T'$of 9 vectors with $(w_1(8), w_2(8))$: (0,0), (0,2), (1,1), (2,1), (1,3), (4,1), (2,2), (3,2), (2,4). Among the 9 values of $w_1(8)$ in T' (0, 0, 1, 2, 1, 4, 2, 3, 2) let us consider the seven values (0, 1, 1, 2, 2, 2, 4) excluding one 0 and one 3, and among the 9 values of $w_2(8)$ in T' (0, 2, 1, 1, 3, 1, 2, 2, 4) let us consider the seven values (0, 1, 1, 2, 2, 2, 4) excluding one 1 and one 3. $T_B^w(4)$ requires at least 5 vectors with $w_{\text{sum}}(4) = 0, 2, 2, 2$ and 4, (i.e., 0 and 4 for one time each, 2 for 3 times) and thus this requirement is satisfied in both cases by the above-mentioned seven values. Thus, T' is sufficient to include $T_B^w(4)$ of the two pieces of *Module(4)* in the stage-A of *Module(8)*. Each of these seven values (0, 1, 1, 2, 2, 2, 4) can be divided as $(0 + 0, 0 + 1, 1 + 0, 1 + 1, 0 + 2, 2 + 0, 2 + 2)$ providing $w_1(4)$ as (0,0,1,1,0,2,2) and $w_2(4)$ as (0,1,0,1,2,0,2). Each set of seven such values of $w_1(4)$ and $w_2(4)$ contains the values (0,1,1,2), which are sufficient for $w_{\text{sum}}(2)$. These four values (0, 1, 1, 2) can be divided as $(0 + 0, 0 + 1, 1 + 0, 1 + 1)$ which are sufficient for $T_B^w(2)$ [Example-8]. From $T_B^w(2)$, we can easily obtain $T_B^{\text{Bool}}(2)$ and $T^{\text{Bool}}(2) = T_B^{\text{Bool}}(2)$ [Lemma-15]. Thus, $T_B^{\text{Bool}}(8)$ can be made to be equal to $T^{\text{Bool}}(8)$. □

Lemma 17. *For* $n = 2^k$ *with* $k > 2$, $T_B^{\text{Bool}}(n)$ *can always be chosen such that* $T^{\text{Bool}}(n) = T_B^{\text{Bool}}(n)$.

Proof: For $k = 4$, we can get $T_B^w(16)$ using Lemma 12. By properly dividing each of $w_1(16)$and $w_2(16)$ into two parts, $T_B^w(16)$ can be made to include $T_B^w(8)$for both *Module(8)* in stage-A [Lemma 14]. Now, for each *Module(8),* $T_B^{\text{Bool}}(8)$ can always be chosen such that $T^{\text{Bool}}(8) = T_B^{\text{Bool}}(8)$ [Lemma 16]. It implies that it is always possible to choose $T_B^{\text{Bool}}(16)$ such that $T^{\text{Bool}}(16) = T_B^{\text{Bool}}(16)$. □

By a similar argument, for $k = 5$, as $T_B^w(32)$ can be made to include both *Module(16)* in stage-A of *Module(32)* and $T^{\text{Bool}}(16) = T_B^{\text{Bool}}(16)$, one can prove $T^{\text{Bool}}(32) = T_B^{\text{Bool}}(32)$.

Using recursion, we can now prove the Lemma.

Lemma 18. *Module(4) requires at least 6 tests.*

Proof: Lemma 14 fails for $n = 4$. That is, $T_B^w(4)$ cannot be made to include $T_B^w(2)$. It is also evident from Example-8 that $w_{\text{sum}}(2) = 1$ appears 2 times in $T_B^w(2)$, but $w_1(4)$ or $w_2(4) = 1$ occurs only one time. Thus, in each of $w_1(4)$ and $w_2(4)$, one additional 1 is needed to include $T_B^w(2)$ of two *Module(2)* of stage-A of *Module(4).* Let us append one additional vector having $w_1(4) = 1$, and $w_2(4) = 1$ with $T_B^w(4)$ to get $T^w(4)$. Thus, $T^w(4)$is given as follows: □

$$T^w(4) = \begin{vmatrix} w_{\min}(4) & w_1(4) & w_2(4) & w_{\text{sum}}(4) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 4 \end{vmatrix}$$

As $T_x^w(4)$ includes $T_B^w(2)$ for both the *Module(2)* in stage-A, and $T^{\text{Bool}}(2) = T_B^{\text{Bool}}(2)$ [Lemma 15], the weight functions in $T_x^w(4)$ can be properly distributed to be sufficient to test the stage-A.

Example 11. From $T_x^w(4)$, by proper distribution of weight functions, we can derive $T^{\text{Bool}}(4)$ as follows:

$$T_x^w(4) = \begin{vmatrix} w_1(4) & w_2(4) \\ 0 & 0 \\ 0 & 2 \\ 2 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{vmatrix} \Rightarrow T_x^w(4) = \begin{vmatrix} w_1(4) & w_2(4) \\ 0+0 & 0+0 \\ 0+0 & 1+1 \\ 1+1 & 0+0 \\ 0+1 & 0+1 \\ 1+0 & 1+0 \\ 1+1 & 1+1 \end{vmatrix} \Rightarrow T^{\text{Bool}}(4) = \begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

Theorem 4. *For $n = 2^k$, Module(n) admits a universal test set. For $k = 1$ and 2, the size of this test set is 4 and 6 respectively, and for $k > 2$, it is $(n^2/8 + 3n/4)$.*

Theorem 5. *Each path in Module(n) is robustly path-delay testable.*

Proof: We will prove the claim recursively. The *Module(n)* is a BIP circuit (i.e., no line exists that reconverges with unequal inversion parities) realizing positive unate functions (Fig. 5). It can be shown that *Module(n)* is irredundant under all multiple stuck-at faults. Let us assume that *Module*($\lceil n/2 \rceil$) and *Module*($\lfloor n/2 \rfloor$) are robustly path-delay testable. In other words, any path from an input to an output of the stage-A of *Module(n)* is robustly testable. In the stage-B of *Module(n)*, the sub-stage-B1 comprises AND or OR gates, inputs to each of which are independent (arriving from two different blocks A and B with no common input). Thus, every path can be sensitized up to the output of the second stage. The sub-stage B-2 consists of AND gates, and the outputs of this sub-stage are OR-ed in the form of a tree in the sub-stage B-3. No two product terms generated at this stage are mutually disjoint. Further, for any particular output, no product term is covered by the any other product term appearing in the Boolean expression of the corresponding function. Thus, for any path starting from a line in the stage-B to any output, non-controlling logic values can be set to all the side inputs of the AND/OR gates along the path. Therefore, *Module(n)* is robustly testable for all path-delay faults if *Module*($\lceil n/2 \rceil$) and *Module*($\lfloor n/2 \rfloor$) are robustly testable. Continuing the argument, it can be inferred that *Module* ($\lceil n/2 \rceil$) {*Module* ($\lfloor n/2 \rfloor$)} will be robustly delay testable, if *Module* ($\lceil (\lceil n/2 \rceil) \rceil$) and *Module* ($\lfloor (\lceil n/2 \rceil) \rfloor$) {*Module* ($\lceil (\lfloor n/2 \rfloor) \rceil$) and *Module*($\lfloor (\lfloor n/2 \rfloor) \rfloor$)} are robustly testable. We can proceed recursively in this fashion to end up at *Module(2)* or *Module(*1) where *Module(2)* consists of simply an OR and an AND gate, and *Module(*1) consists of a single line. Obviously, all the paths in *Module(2)* and *Module(*1) are robust testable. Thus, all paths in *Module(n)* are robustly path-delay testable.

6. Synthesis of General Symmetric Functions

6.1. Consecutive Symmetric Functions

To synthesize a consecutive symmetric function that is not unate, we use the result stated in Theorem 1 [3] that $S^n(a_l - a_r) = S^n(a_l - n)\overline{S^n}(a_{r+1} - n) = u_l(n).\,\overline{u}_{r+1}(n)$. The unate functions $u_l(n)$ and $u_{r+1}(n)$ are produced by *Module(n)*. The complete circuit is shown in Fig. 14(a).

Example 12. $S^6(3, 4)$ is realized as $S^6(3, 4) = S^6(3 - 6).\overline{S}^6(5 - 6) = u_3(3) \cdot \overline{u}_5(5)$. It is shown in Fig. 14(b).

Theorem 6. *The above implementation of any consecutive symmetric function $S^n(a_l - a_r)$, $(a_l \neq a_r)$, is robustly path-delay testable.*

Proof: Follows from Theorem 4 and the results in [3]. □

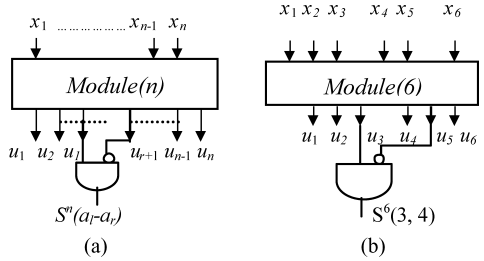

Fig. 14. Realization of (a) $S^n(a_l - a_r)$ (b) $S^n(3, 4)$.

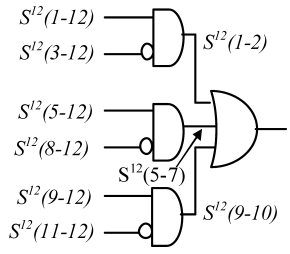

Fig. 15. Testable circuit realizing $S^{12}(1, 2, 5, 6, 7, 9, 10)$.

Table 1. Comparative features of *Module* (n)

	As in [6]	As in [8]	Proposed design
Cost(# 2-input gates)	$n(n-1)$	$n(n-1)$	$=(n^2+n\log n-n)/2$ for $n=2^k$ $=C(p)+C(n-p)+(n-p)(3p-n+1)$ for $2^{k-1}<n\leq 2^k$ where $p=2^{k-1}$
Delay			
Min.	n	n	$\lceil \log n \rceil$
Max.	$2n\cdot(n-1)$	$2n\cdot(n-1)$	$=\lceil \log n\rceil\ (\lceil \log n\rceil +3)/2-2$ for $n>3p/2$ $=\lceil \log n\rceil\ (\lceil \log n\rceil +3)/2-3$ for $n=3p/2$ $=\lceil \log n\rceil\ (\lceil \log n\rceil +1)/2+\lceil \log(n-p+1)\rceil -2$ for $n<3p/2$. where $2^{k-1}<n<2^k$ and $p=2^{k-1}$
Path-delay testability	Not testable	Robustly testable	Robustly testable

Table 2. Cost and delay of *Module*(n)

	# 2-input gates		Delay			
	as in [6]		as in [6] and [8]		Proposed method	
n	& [8]	Proposed method	Min	Max	Min	Max
1	0	0	0	0	0	0
2	2	2	2	3	1	1
3	6	6	3	5	2	3
4	12	10	4	7	2	3
5	20	18	5	9	3	6
6	30	24	6	11	3	6
7	42	34	7	13	3	7
8	56	40	8	15	3	7
9	72	56	9	17	4	10
10	90	66	10	19	4	10
11	110	82	11	21	4	10
12	132	90	12	23	4	10
13	156	112	13	25	4	11
14	182	124	14	27	4	11
15	210	144	15	29	4	12
16	240	152	16	31	4	12

Table 3. Comparison of area and delay for benchmark circuits

			Area			Delay		
Circuit	# Inputs	# outputs	Original circuit	After optimization	Proposed tech.	Original circuit	After optimization	Proposed tech.
sym9	9	1	383	290	84	13	14	8
sym10	10	1	286	167	98	15	16	9
rd53	5	3	85	48	45	11	12	7
rd73	7	3	220	110	84	11	13	8
rd84	8	4	308	132	101	15	11	9

6.2. Nonconsecutive Symmetric Functions

To synthesize a nonconsecutive symmetric function for robust path-delay testability, it is first expressed as a union of several maximal consecutive symmetric functions, and then each of the constituent consecutive symmetric functions is realized by combining the appropriate outputs of *Module(n)*, via unate decomposition. Finally, they are ORed together. It is shown in [3] that the overall circuit based on such decomposition is robustly path-delay fault testable. Determination of a complete test set for detecting all path-delay faults needs further analysis of the circuit. Implementation of a nonconsecutive symmetric function (Example 1: $S^{12}(1, 2, 5, 6, 7, 9, 10)$) is shown in Fig. 15.

Table 4. Cost of general symmetric functions

	Number of gate inputs				Number of paths			
Functions $S^n(a_l - a_r)$	TCAD95 [2]	TCAD00 [3]	VLSI02 [9]	Proposed method	TCAD95 [2]	TCAD00 [3]	VLSI02 [9]	Proposed method
S^{10}(5,6)	2354	1296	130	97	2100	1034	420	392
S^{11}(6,7)	4556	2433	174	111	4092	1957	1278	466
S^{11}(3–7)	2147	1108	174	107	1815	916	1066	346
S^{11}(3,4)	3137	1675	174	105	2805	1365	266	360
S^{11}(5,6)	5082	2740	174	110	4620	2220	676	448
S^{11}(4,5)	4556	2433	174	109	4092	1957	434	512
S^{12}(7,8)	8318	4330	190	124	7524	3548	1492	504
S^{12}(4–8)	4455	2340	190	123	3960	1960	1124	432
S^{12}(6,7)	10430	5463	190	123	9504	4469	1308	552
S^{12}(5,6)	10430	5463	190	126	9504	4469	692	528
S^{13}(7,8)	20165	10261	218	147	18447	8451	2362	770
S^{13}(4–8)	10584	5336	218	144	9295	4494	1814	614
S^{13}(6,7)	22308	11518	218	148	20592	9530	2094	848
S^{13}(5,6)	20165	10261	218	145	18447	8451	944	786
S^{14}(8,9)	37039	18596	234	165	34034	15540	3560	1176
S^{14}(5–9)	22022	11262	234	166	20020	9578	1468	1112
S^{14}(5,6)	37039	18596	234	165	34034	15540	894	1020
S^{14}(6,7)	45476	22877	234	166	42042	19081	2220	1056
S^{15}(5–9)	50052	24671	286	183	45045	21085	2559	1292
S^{15}(5,6)	65067	32312	286	179	60060	27354	1170	1188
S^{15}(7,8)	96525	47950	286	182	90090	40362	3189	1382

7. Experimental Results

We compare the hardware cost and delay of *Module(n)* with the earlier designs reported in [6, 8] in Table 1 and Table 2. Both the parameters are favorably reduced in the new design. For general consecutive symmetric functions, we compare the hardware cost and the number of paths with those in [2, 3, 9] (Table 4). The results show a significant reduction in circuit cost compared to those in [2, 3]. While the earlier methods use a fixed number of logic levels, for instance, at most 4 [2], or at most 5 [3], the proposed method reduces the logic significantly at the cost of increasing the number of levels. However, the number of paths in the proposed design reduces drastically compared to that in [2, 3, 9]. Table 3 depicts results on some benchmark circuits realizing symmetric functions taken from MCNC benchmarks [20]. For each benchmark circuit, we compute the area and delay of both the original and optimized circuits. These circuits are not path-delay testable. Moreover, except *9sym,* no other circuit has two-level path-delay testable realization. The proposed implementation technique using hierarchical modules ensures path-delay fault testability for all these circuits and yields lesser area and (max) delay compared to those of the original implementations. To calculate the area, some gates are replaced by equivalent gates. Say, for example an AND-OR realization is replaced by NAND-NAND. We have used the *SIS* tool [10] and *mcnc.genlib* library to estimate area for comparison.

8. Conclusion

The proposed procedure for implementing a symmetric Boolean function using hierarchical modules is simple and it guarantees robust testability of all path-delay faults. Multiple symmetric functions of n variables can be implemented by using only one block of *Module(n)* and some additional logic. A test set of size $O(n^2)$ for detecting all single stuck-at faults in the module can be easily derived. The number of paths in the circuit is reduced significantly compared to the earlier designs, and hence the time needed for delay test generation and test application is likely to reduce proportionately. Although the design is shown to be robustly path-delay testable, determination of an optimum test sequence for detecting all path-delay faults in the circuit remains an open problem.

References

1. D.L. Dietmeyer, "Generating minimal covers of symmetric function," *IEEE TCAD,* Vol. 12, No. 5, pp. 710–713, 1993.

2. W. Ke and P.R. Menon, "Delay-testable implementations of 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*, McGraw-Hill, New York, 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. S.L. Hurst, "Digital summation threshold logic gates: A new circuit element," *IEE Proc*., Vol. 120, No. 11, pp. 1301–1307, 1973.
7. 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.
8. H. Rahaman, D.K. Das, and B.B. Bhattacharya, "A simple delay-testable design of digital summation threshold logic (DSTL) array," in *Proc. 5th International Workshop on Boolean Problems,* Freiberg, Germany, September 2002.
9. H. Rahaman, D.K. Das, and B.B. Bhattacharya, "A new synthesis of symmetric functions," in *Proc. Int. Conf. ASP-DAC/VLSI Design,* 2002, pp. 160–165.
10. E.M. Sentovich, et al., "SIS: A sequential system for sequential circuit synthesis," *Technical Report UCB/ERL m92/41. Electronic Research Laboratory,* University of California, Berkeley, May 1992.
11. M. Perkowski, P. Kerntopf, A.Buller, M.C.-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L.Jozwiak, A. Coppola, and B. Massey, "Regularity and symmetry as a base for efficient realization of reversible logic circuits," *Manuscript*, 2001.
12. P. Picton, "Modified fredkin gates in logic design," *Microelectronics Journal,* Vol. 25, pp. 437–441, 1994.
13. G.L. Smith, "Model for delay faults based upon paths," in *Proc. Int. Test Conf.*, 1985, pp. 342– 349.
14. C.J. Lin and S.M. Reddy, "On delay fault testing in logic circuits," *IEEE Trans. CAD*, Vol. CAD-6, pp. 694–703, Sept. 1987.
15. R. Betancourt, "Derivation of minimum test sets for unate logic circuits," *IEEE Trans. Comput.*, Vol. C-20, pp. 1264–1269, 1971.
16. S. B. Akers, "Universal test sets for logic networks," *IEEE Trans. Comput*., Vol. C-22, pp. 835–839, 1973.
17. S.M. Reddy, "Complete test set for logic functions," *IEEE Trans. Comput*., Vol. C-22, pp. 1016–1020, Nov. 1973.
18. U. Sparmann, et al., "Minimal delay test for unate gate networks," in *Proc. Asian Test Symp.*, pp. 10–16, 1997.
19. H. Kim and J.P. Hayes, "Realization-independent ATPG for designs with unimplemented blocks," *IEEE Trans. CAD,* Vol. 20, pp. 290–306, Feb. 2001.
20. S. Yang, "Logic synthesis and optimization benchmarks guide," *Technical Report 1991-IWLS-UG-Saeyang, Microelectronics Center of North Carolina.*

Hafizur Rahaman received the Bachelor of Electrical Engineering degree from the Bengal Engineering College, Calcutta University, India in 1986, the Master degree in Electrical Engineering, and the Ph.D. degree in computer science and engineering from the Jadavpur University, Calcutta, India in 1988 and 2003 respectively. He is currently chairing the Department of Information Technology, Bengal Engineering and Science University, Shibpur, India. His research interest includes logic synthesis and testing of VLSI circuits, fault-tolerant computing, and quantum computing. He has published several papers in well-known international journals and in reputed conference proceedings. He served in the Organizing Committee of the International Conference on VLSI Design in 2000 and 2005, and as the Registration Chair of the 2005 Asian Test Symposium (ATS), in Kolkata.

Debesh K. Das received the Bachelor and Master degrees in electronics and telecommunication engineering, and the Ph.D. degree in computer science and engineering all from the Jadavpur University, Calcutta, India in 1982, 1984, and 1997 respectively. He is currently full professor at the Department of Computer Science and Engineering, Jadavpur University. He visited the University of Potsdam, Germany, the Asian Institute of Technology, Bangkok, the Abdus Salam International Center for Theoretical Physics, Trieste, Italy, and the Nara Institute of Science and Technology, Japan. His research work primarily focuses on logic synthesis and testing of VLSI circuits and fault-tolerant computing. He has published numerous papers in archival journals and refereed international conference proceedings. He served in the Organizing Committee of the International Conference on VLSI Design in 2000 and 2005. He also served as the Organizing Chair of the 2005 Asian Test Symposium (ATS).

Bhargab B. Bhattacharya received the B.Sc. degree in physics from the Presidency College, Calcutta, the B.Tech. and M.Tech. degrees in radiophysics and electronics, and the Ph.D. degree in computer science all from the University of Calcutta, India. Since 1982, he has been on the faculty of the Indian Statistical Institute, Calcutta, where currently he is full professor. He held visiting professorship at the University of Nebraska-Lincoln, USA, and at the University of Potsdam, Germany. In 2005, he visited the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur as VSNL Chair Professor. His research and teaching interest includes logic synthesis, testing and physical design of VLSI circuits, nanotechnology, graph and geometric algorithms, and image processing architecture. He is author of more than 180 papers published in archival journals and refereed conference proceedings, and inventor of 8 United States patents. Currently, he is collaborating with Intel Corporation, USA, and IRISA, France, for development of image processing hardware, chip layout analysis and design, and reconfigurable parallel computing tools.

Dr. Bhattacharya is a Fellow of the *Indian National Academy of Engineering*, a Fellow of the *National Academy of Sciences*, India, and a recipient of the *VASVIK Award for Electronic Sciences and Technology*. He is on the editorial board of the *Journal of Circuits, Systems, and Computers* (World Scientific, Singapore), and the J*ournal of Electronic Testing—Theory and Applications* (Springer).