Mapping Symmetric Functions to Hierarchical Modules for Path-Delay Fault Testability

Hafizur Rahaman
IIIT-C
WB University of Technology
Kolkata –700 091, India
rahaman_h@hotmail.com

Debesh K. Das
Dept. Computer Sci. & Eng.
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

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. General symmetric functions are then realized following a unate decomposition method. The synthesis procedure guarantees complete and 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 in earlier designs. Results on circuit area and delay for a few benchmark examples are also reported.

1. Introduction

Synthesis and testing of symmetric Boolean functions received lot of interest in the past [1 - 4, 7, 9]. 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]. Such an array can be used directly for synthesizing unate symmetric functions. Non-unate symmetric functions can then be synthesized by the method proposed in [3]. In the proposed work, we change the internal structure of the module such that it can be used to synthesize any general symmetric function with 100% robust path-delay fault testability. The cost of the circuit is similar to that of an earlier design [9]. However, the number of paths in the circuit reduces drastically compared to those in [2, 3, 8, 9].

2. Preliminaries

Let $f(x_1, x_2, ..., x_n)$ 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 *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 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) $A = (a_i, .., a_j, ..., a_k)$, where $A \subseteq \{0, 1, 2, ..., n\}$; 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 $S^n(A) = S^n(A_1) + S^n(A_2) + ... + S^n(A_m)$, such that m is minimum and $\forall i, j, 1 \leq i, j \leq m, A_i \cap A_j = \emptyset$, whenever $i \neq j$.

Example 1: The symmetric function $S^{12}(1,2,5,6,7,9,10)$ can be expressed as $S^{12}(1\text{-}2) + S^{12}(5\text{-}7) + S^{12}(9\text{-}10)$, 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. 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 $(n\text{-}1)$ functions: $S^n(1\text{-}n)$, $S^n(2\text{-}n)$, $S^n(3\text{-}n)$,...., $S^n((n\text{-}1) - n)$. We express $S^n(n)$ as $u_n(n)$, and $S^n(a_l\text{-}a_r)$ as $u_l(n)$ for $1 \leq l \leq (n\text{-}1)$.

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

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

3. Synthesis of unate symmetric functions

Unate symmetric functions can be synthesized by a DSTL array [6]. To achieve path-delay fault testability, the above design is modified in [8]. A synthesis technique for implementing symmetric functions was proposed in [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. 1). Each output u_i, implements a unate symmetric function as described below (where $\sum$ denotes Boolean OR operation):

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

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 $n = 2^k$

We first assume $n = 2^k$. For $k = 1$, the design is same as that in [6], and we redraw it as in Fig. 2. For $k > 1$, the *Module(n)* consists of three stages as shown in Fig. 3.

First-stage: This consists of two blocks – Block A and Block B (Fig. 4), each of which is identical to *Module(n/2)*. The inputs to the Block A (B) 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 $a_1, a_2, a_3, \ldots, a_{n/2}$ and $b_1, b_2, b_3, \ldots, b_{n/2}$. 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.

Example 2: The first-stage for $n = 4$ is shown in Fig. 5.

Second stage: This consists of $n/2$ OR gates and $n/2$ AND-OR gate pairs (Fig. 6) with n inputs. The n outputs are given by $a_1+b_1, a_1b_1, a_2+b_2, a_2b_2, a_3+b_3, a_3b_3, \ldots a_{n/2-2}+b_{n/2-2}, a_{n/2-2}b_{n/2-2}, a_{n/2-1}+b_{n/2-1}, a_{n/2-1}b_{n/2-1}, a_{n/2}+b_{n/2}, a_{n/2}b_{n/2}$. The 1st ($n^{th}$) output represents $u_1(n)$ (resp. $u_n(n)$).

Example 3: Fig. 7a(7b) shows the 2nd stage for $n = 4$ (8).

Third stage: The algebraic expressions for output functions of the 3rd stage $f(z_1), f(z_2), f(z_3), \ldots, f(z_i), \ldots f(z_n)$, in terms of the output functions of the second stage are given by:

$f(z_1) = a_1 + b_1$
$f(z_2) = a_1b_1 + (a_2 + b_2)$
$f(z_3) = a_1b_1(a_2 + b_2) + (a_3 + b_3)$
$f(z_4) = a_1b_1(a_3 + b_3) + a_2b_2 + (a_4 + b_4)$
………………
$f(z_{n/2}) = a_1b_1(a_{n/2-1}+b_{n/2-1}) + a_2b_2(a_{n/2-2} + b_{n/2-2}) + \ldots + a_{n/4}b_{n/4} + (a_{n/2} + b_{n/2})$
$f(z_{n/2+1}) = a_1b_1(a_{n/2}+b_{n/2}) + a_2b_2(a_{n/2-1} + b_{n/2-1}) + \ldots + a_{n/4}b_{n/4}(a_{n/4+1} + b_{n/2+1})$
$f(z_{n-2}) = a_{n/2-2}b_{n/2-2}(a_{n/2} + b_{n/2}) + a_{n/2-1}b_{n/2-1}$
$f(z_{n-1}) = a_{n/2-1}b_{n/2-1}(a_{n/2} + b_{n/2})$
$f(z_n) = a_{n/2}b_{n/2}$

More specifically,

$$f(z_i) = a_1b_1(a_{i-1} + b_{i-1}) + a_2b_2(a_{i-2} + b_{i-2}) + a_3b_3(a_{i-3} + b_{i-3}) + \ldots\ldots + a_{(i-1)/2}b_{(i-1)/2}(a_{(i+1)/2} + b_{(i+1)/2}) + \beta a_{i/2}b_{i/2} + a_i + b_i$$

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

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

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

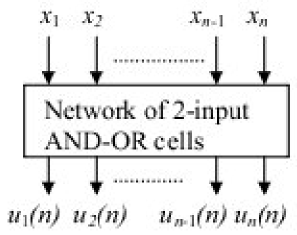
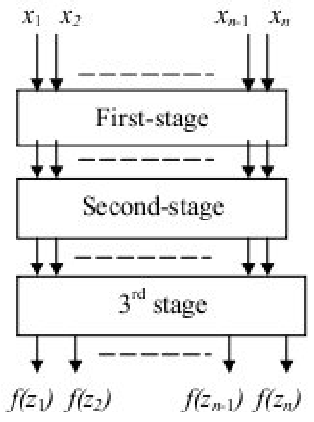

x_1 x_2 x_{n-1} x_n
Network of 2-input AND-OR cells
$u_1(n)$ $u_2(n)$ $u_{n-1}(n)$ $u_n(n)$

Fig. 1: *Module(n)*


x_1 x_2 x_{n-1} x_n
First-stage
Second-stage
3rd stage
$f(z_1)$ $f(z_2)$ $f(z_{n-1})$ $f(z_n)$

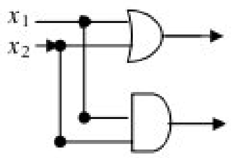


Fig. 2: *Module(2)* **Fig. 3:** The Structure of *Module(n)*

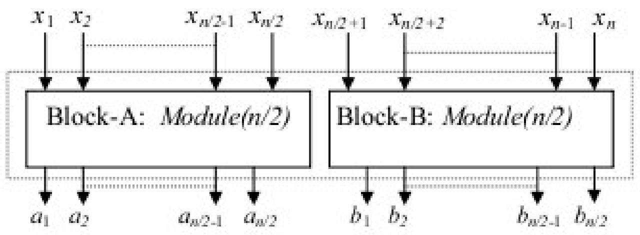

x_1 x_2 $x_{n/2-1}$ $x_{n/2}$ $x_{n/2+1}$ $x_{n/2+2}$ x_{n-1} x_n
Block-A: Module(n/2)
Block-B: Module(n/2)
a_1 a_2 $a_{n/2-1}$ $a_{n/2}$ b_1 b_2 $b_{n/2-1}$ $b_{n/2}$

Fig. 4: 1st stage of *Module(n)*

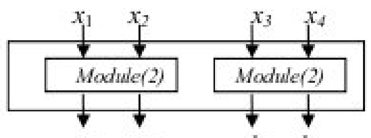


Fig. 5: 1st stage of *Module (4)*

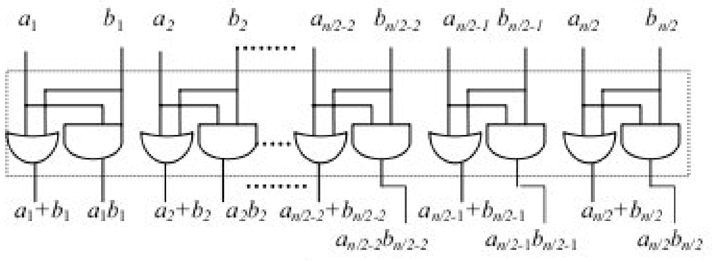

a_1 b_1 a_2 b_2 $a_{n/2-2}$ $b_{n/2-2}$ $a_{n/2-1}$ $b_{n/2-1}$ $a_{n/2}$ $b_{n/2}$
a_1+b_1 a_1b_1 a_2+b_2 a_2b_2 $a_{n/2-2}+b_{n/2-2}$ $a_{n/2-2}b_{n/2-2}$ $a_{n/2-1}+b_{n/2-1}$ $a_{n/2-1}b_{n/2-1}$ $a_{n/2}+b_{n/2}$ $a_{n/2}b_{n/2}$

Fig. 6: 2nd stage of *Module(n)*

Example 4: The 3rd stage for n = 4 (8) is shown in Fig. 8a (8b).

Example 5: Module(4) [*Module(8)*] is shown in Fig. 9a [9b].

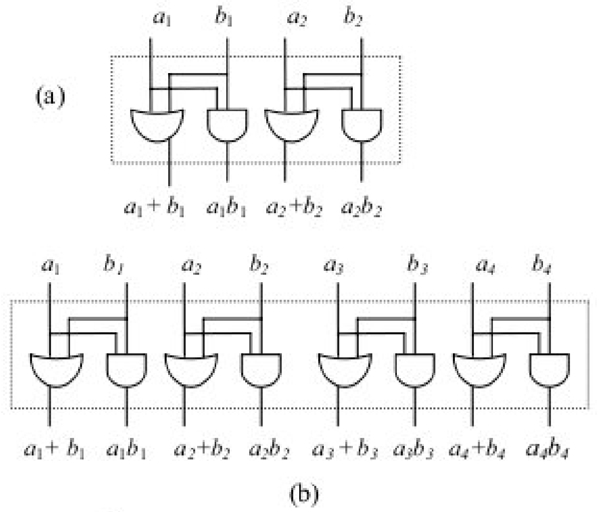

(a)
a_1 b_1 a_2 b_2
a_1+b_1 a_1b_1 a_2+b_2 a_2b_2
a_1 b_1 a_2 b_2 a_3 b_3 a_4 b_4
a_1+b_1 a_1b_1 a_2+b_2 a_2b_2 a_3+b_3 a_3b_3 a_4+b_4 a_4b_4
(b)

Fig. 7: 2nd stage of *Module(n)* for (a) $n = 4$ (b) $n = 8$

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 $\begin{bmatrix} n/2 \\ j \end{bmatrix}$ and $\begin{bmatrix} n/2 \\ k \end{bmatrix}$ 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 $k > j$
Proof: Follows from Lemmas 1 and 2.

Lemma 4: The minimum s-o-p expression for $\{a_j\, b_k\}$ has $\begin{bmatrix} n/2 \\ j \end{bmatrix} * \begin{bmatrix} n/2 \\ k \end{bmatrix}$ 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 $1 \le i_1, j_1, i_2, j_2 \le n/2$ 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 realizes *Module(n)*.

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

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

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

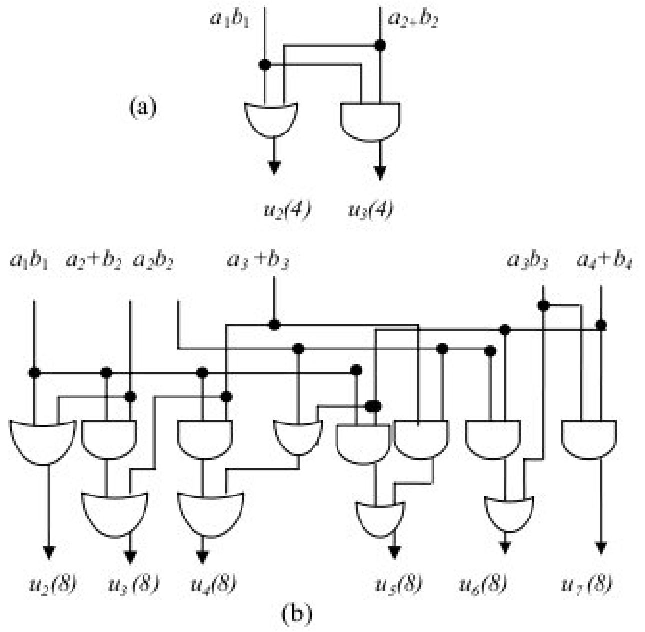

a_1b_1 a_2+b_2
(a)
$u_2(4)$ $u_3(4)$
a_1b_1 a_2+b_2 a_2b_2 a_3+b_3 a_3b_3 a_4+b_4
$u_2(8)$ $u_3(8)$ $u_4(8)$ $u_5(8)$ $u_6(8)$ $u_7(8)$
(b)

Fig. 8: Third stage *of Module(n)* for (a) $n = 4$ (b) $n = 8$

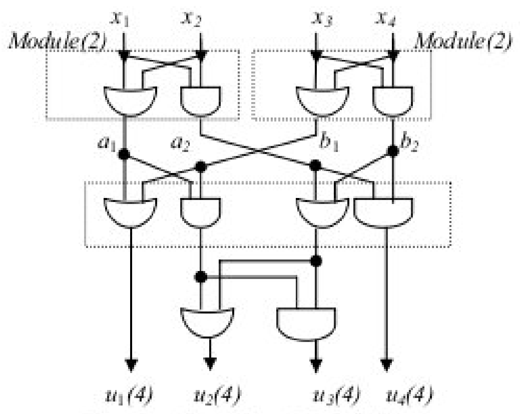


Fig. 9a: Complete *Module(4)*

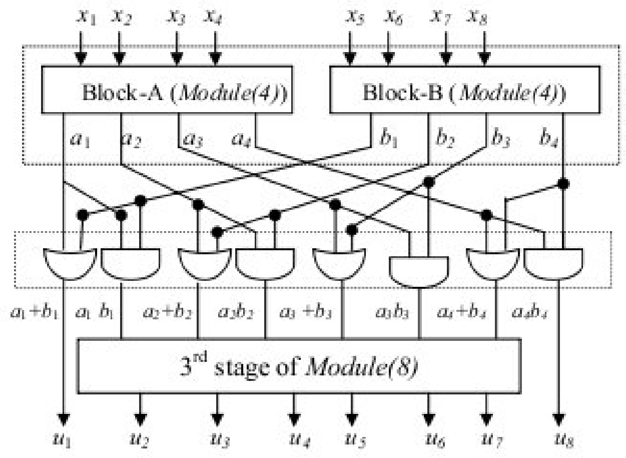
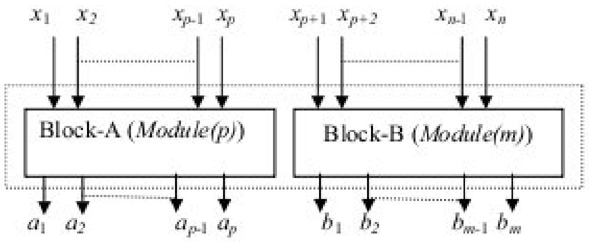

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
Block-A (Module(4))
Block-B (Module(4))
a_1 a_2 a_3 a_4 b_1 b_2 b_3 b_4
a_1+b_1 a_1b_1 a_2+b_2 a_2b_2 a_3+b_3 a_3b_3 a_4+b_4 a_4b_4
3rd stage of Module(8)
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8

Fig. 9b: *Module(8)*

3.2 Designing *Module(n)* for $2^{k-1} < n < 2^k$

For $2^{k-1} < n < 2^k$, where k is an integer, *Module(n)* will have three stages, similar to the case when $n = 2^k$. Let us assume that $p + m = n$, where $p = 2^{k-1}$.

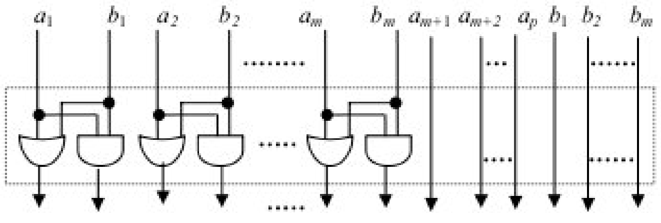
First stage: This consists of two parts: *Module(p)* and *Module(m)* (Fig. 10). The outputs of first stage feed the second stage.



x_1 x_2 x_{p-1} x_p
x_{p+1} x_{p+2} x_{n-1} x_n
Block-A (*Module(p)*)
Block-B (*Module(m)*)
a_1 a_2 a_{p-1} a_p
b_1 b_2 b_{m-1} b_m

Fig. 10: 1st stage of *Module(n)* for $2^{k-1} < n < 2^k$, with $p = 2^{k-1}$ and $p + m = n$

Second-stage: The 2nd stage of *Module(n)* (Fig. 11) includes *m* OR-AND pair gates arranged in a block. The outputs of the 2nd stage realizing (*2m*) functions are given by $a_1+ b_1$, a_1b_1, $a_2+ b_2$, a_2b_2, ...$a_{m-1}+ b_{m-1}$, $a_{m-1}b_{m-1}$, $a_m+ b_m$, a_mb_m, the 1*st* output $(a_1 + b_1)$ being equal to $u_1(n)$. The lines corresponding to functions a_{m+1}, ...,a_p, b_1, b_2, b_3, ..., b_m just pass through the 2nd stage and appear as outputs.



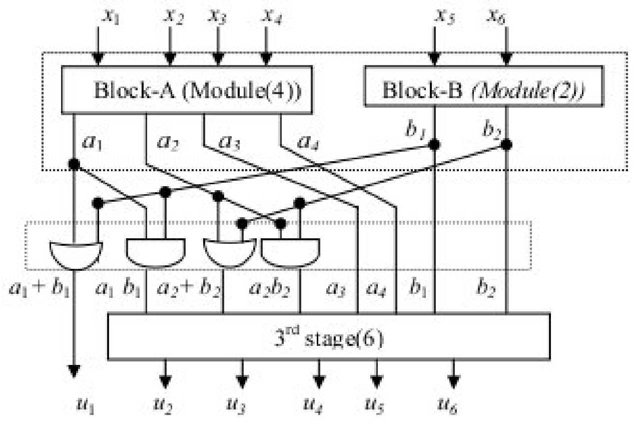
a_1 b_1 a_2 b_2 a_m b_m a_{m+1} a_{m+2} a_p b_1 b_2 b_m

Fig. 11: 2nd stage of *Module(n)* for $2^{k-1} < n < 2^k$, with $p = 2^{k-1}$ and $p + m = n$

Third stage: This is obtained by removing some gates and lines from the third stage for *Module*(2^k), and changing a few gate inputs. The third stage for *Module*(2^k) has input lines realizing a_1+b_1, a_1b_1, a_2+b_2, a_2b_2, ...a_m+b_m, a_mb_m, $a_{m+1} +b_{m+1}$, $a_{m+1}b_{m+1}$,.........., $a_{p-1}+b_{p-1}$, $a_{p-1}b_{p-1}$, a_p+b_p, a_pb_p. To obtain the design for $n < 2^k$, we replace the inputs $a_{m+1} +b_{m+1}$, $a_{m+2}+b_{m+2}$........, $a_{p-1}+b_{p-1}$, a_p+b_p by a_{m+1}, a_{m+2},......., a_{p-1}, a_p respectively. The lines realizing $a_{m+1}b_{m+1}$, $a_{m+2}b_{m+2}$,......., $a_{p-1}b_{p-1}$, a_pb_p and the gates fed by them are removed from the design. Further, the input $a_i b_i$ that is ANDed with a_j+b_j for $1 \le i \le m$ and $m < j \le p$, is replaced by b_j.

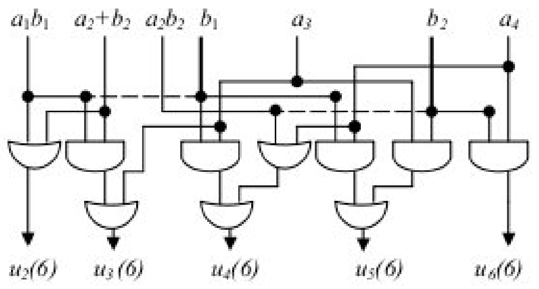
Example 6: The complete module for $n = 6$ is shown in Fig. 12, where the third stage is shown as a block.

Example 7: The 3rd stage of *Module(6)* (Fig. 13) is obtained by removing some gates and lines from the 3rd stage of *Module(8)* (Fig. 8b). The dotted lines, which are present in *Module(8)*, are removed in *Module(6)*. The bold lines in *Module(6)* show the changes in input connections.



x_1 x_2 x_3 x_4 x_5 x_6
Block-A (Module(4))
Block-B *(Module(2))*
a_1 a_2 a_3 a_4 b_1 b_2
a_1+b_1 a_1b_1 a_2+b_2 a_2b_2 a_3 a_4 b_1 b_2
3rd stage(6)
u_1 u_2 u_3 u_4 u_5 u_6

Fig. 12: *Module (6)*



a_1b_1 a_2+b_2 a_2b_2 b_1 a_3 b_2 a_4
$u_2(6)$ $u_3(6)$ $u_4(6)$ $u_5(6)$ $u_6(6)$

Fig. 13: Third stage of *Module(6)*

4. Circuit cost, delay, and testability

The new 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 for fast evaluation of inputs.

Hardware cost

Let *C(n)* denote the number of 2-input gates in *Module(n).* For $n = 2^k$, $C(n) = 2C(n/2) + n + n(n-2)/4 = \{n^2/2 + n/2\{\log n - 1\} = n/2\{n + k - 1\}$. For $n \neq 2^k$, $C(n) \le \lceil n/2 \rceil \{n + \lceil \log n \rceil\}$. The results for $2 \le n \le 16$ is shown in Table 2.

Circuit delay

We assume unit gate delay through a 2-input gate. For $n = 2^k$, the minimum delay through the circuit is *k*, and the maximum delay $D_{max}(n)$ is $[\{(k^2 + k\}/2]$. Dealy values for $2 \le n \le 16$ are shown in Table 2.

Effects on testability

The modules based on DSTL array [6] or its modification [9] are not fully path-delay testable. The proposed design circumvents this problem.

Theorem 3: Each path in *Module(n)* is robustly delay testable.

Proof: We will prove the claim recursively. For simplicity, we assume that $n = 2^k$, $k \geq 1$. *Module(n)* is a unate circuit (i.e., no line exists that reconverges with unequal inversion parities). It can be shown that *Module(n)* is irredundant under all multiple stuck-at faults. Let us assume that *Module(n/2)* is fully path-delay testable. In other words, a path from any input to any output of the first stage is robustly testable. In the second stage, the first level 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. In the third stage, the first level consists of AND gates, the outputs of which are OR-ed in the form of a tree. 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 third stage 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 fully robust testable for all path-delay faults. Similarly, *Module(n/2)* will be robustly delay testable, if *Module(n/4)* is robustly testable. We can proceed recursively in this fashion to end up at *Module(2)* that consists of simply an OR and an AND gate, paths in which are obviously robust testable. Thus, all paths in *Module(n)* are robustly delay testable.

5. Synthesis of general symmetric functions

5.1 Consecutive symmetric functions

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

Example 8: $S^6(3, 4)$ 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. 14b.

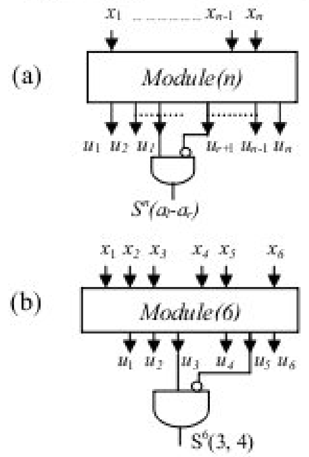


Fig. 14: Realization of (a) $S^n(a_l\text{-}\ a_r)$ (b) $S^n(3, 4)$

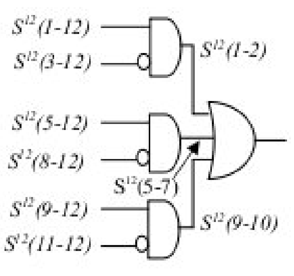


Fig. 15: Testable circuit realizing $S^{12}(1,2,5,6,7,9,10)$

Theorem 4: The above implementation of any consecutive symmetric function $S^n(a_l\text{ - }a_r)$, $(a_l \neq a_r)$, is robustly path-delay testable.

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

5.2 Nonconsecutive symmetric functions

To synthesize a nonconsecutive symmetric function for 100% 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 OR-ed together. It is shown in [3] that the overall circuit based on such decomposition is robustly path-delay fault testable. Implementation of a nonconsecutive symmetric function (Example 1: $S^{12}(1,2,5,6,7,9,10)$) is shown in Fig. 15.

6. Experimental results

We compare the hardware cost and delay of *Module(n)* with 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, and in turn, testing time in the proposed design reduces drastically compared to that in [2, 3, 9]. Table 3 depicts results on some benchmark circuits realizing symmetric functions. These circuits are not path-delay testable. Moreover, except *9sym* no other circuit has two-level 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. We have used the SIS tool [10] and mcnc.genlib library to estimate area for comparison.

7. Conclusion

The proposed procedure for implementing symmetric Boolean functions using hierarchical modules is simple, and guarantees robust testability of 100% path-delay faults. Multiple symmetric functions of *n* variables can be implemented by using only one block of *Module(n)* and some additional logic. The number of paths in the circuit is reduced significantly compared to earlier designs, and hence time needed for delay test generation and test application is likely to reduce proportionately. Determination of an optimum test sequence for path-delay faults in the circuit so designed is an open problem.

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 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*. 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. 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", *Proc. of the 5th International Workshop on Boolean Problems,* September 2002, Freiberg, Germany.
9. H. Rahaman, D. K. Das, and B. B. Bhattacharya, " A new synthesis of symmetric functions," *Proc. of the Int. Conf. ASP-DAC/VLSI Design,* pp. 160-165, January 2002.
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.

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)*	$= \{n/2\,(n+k-1)\}$, for $n = 2^k$ $\le \lceil n/2 \rceil \{n + \lceil \log n \rceil\}$ for $n \ne 2^k$
Delay	Min. Max.	n $2n.(n-1)$	n $2n.(n-1)$	$\lceil \log n \rceil$ $\{\lceil \log n \rceil^2 + \lceil \log n \rceil\}/2$
Path-delay testability		Not testable	Robustly testable	Robustly testable

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

n	# 2-input gates		Delay			
	as in [6] & [8]	Proposed method	as in [6] and [8]		Proposed method	
			Min	Max	Min	Max
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	6
8	56	40	8	15	3	6
9	72	56	9	17	4	10
10	90	66	10	19	4	10
11	110	78	11	21	4	10
12	132	90	12	23	4	10
13	156	112	13	25	4	10
14	182	124	14	27	4	10
15	210	144	15	29	4	10
16	240	152	16	31	4	10

Table 3: Comparison of area and delay for benchmark circuits

Circuit	#inputs	#outputs	Area		Delay	
			original circuit	proposed tech.	original circuit	proposed tech.
sym9	9	1	202	84	13	9
sym10	10	1	159	99	15	10
rd53	5	3	50	47	11	7
rd73	7	3	93	88	11	8
rd84	8	4	228	112	15	9

Table 4: Cost of general symmetric functions

Functions	Number of gate inputs				Number of paths			
$S^n(a_i\text{-}a_r)$	as in [2]	as in [3]	as in[9]	Proposed method	as in [2]	as in [3]	as in [9]	Proposed method
$S^{10}(5,6)$	2354	1296	130	134	2100	1034	420	320
$S^{11}(6,7)$	4556	2433	174	158	4092	1957	1278	430
$S^{11}(3\text{-}7)$	2147	1108	174	158	1815	916	1066	294
$S^{11}(3,4)$	3137	1675	174	158	2805	1365	266	270
$S^{11}(5,6)$	5082	2740	174	158	4620	2220	676	430
$S^{11}(4,5)$	4556	2433	174	158	4092	1957	434	358
$S^{12}(7,8)$	8318	4330	190	182	7524	3548	1492	520
$S^{12}(4\text{-}8)$	4455	2340	190	182	3960	1960	1124	384
$S^{12}(6,7)$	10430	5463	190	182	9504	4469	1308	494
$S^{12}(5,6)$	10430	5463	190	182	9504	4469	692	422
$S^{13}(7,8)$	20165	10261	218	226	18447	8451	2362	622
$S^{13}(4\text{-}8)$	10584	5336	218	226	9295	4494	1814	500
$S^{13}(6,7)$	22308	11518	218	226	20592	9530	2094	590
$S^{13}(5,6)$	20165	10261	218	226	18447	8451	944	550
$S^{14}(8,9)$	37039	18596	234	250	34034	15540	3560	826
$S^{14}(5\text{-}9)$	22022	11262	234	250	20020	9578	1468	700
$S^{14}(5,6)$	37039	18596	234	250	34034	15540	894	700
$S^{14}(6,7)$	45476	22877	234	250	42042	19081	2220	770
$S^{15}(5\text{-}9)$	50052	24671	286	290	45045	21085	2559	750
$S^{15}(5,6)$	65067	32312	286	290	60060	27354	1170	750
$S^{15}(7,8)$	96525	47950	286	290	90090	40362	3189	855