Short Papers

Synthesis of Symmetric Functions for Path-Delay Fault Testability

Susanta Chakrabarti, Sandip Das, Debesh K. Das, and Bhargab B. Bhattacharya

***Abstract*—A new technique of synthesizing totally symmetric Boolean functions is presented that achieves complete robust path-delay fault testability. We show that every consecutive symmetric function can be expressed as a logical composition (e.g., AND, NOR) of two unate symmetric functions, and the resulting composite circuit can be made robustly path-delay fault testable, if the constituent unate functions are synthesized as two-level irredundant circuits. Nonconsecutive symmetric functions can also be synthesized by decomposing them into a set of consecutive symmetric functions. The circuit cost of the proposed design can further be reduced by a novel algebraic factorization technique based on some combinatorial clues. The overall synthesis guarantees complete robust path-delay fault testability, and can be completed in linear time. The results shows that the proposed method ensures a significant reduction in hardware, as well as in the number of paths, which in turn, reduces testing time, as compared to those of the best-known earlier methods.**

***Index Terms*—Delay faults, symmetric Boolean functions, synthesis-for-testability.**

I. Introduction

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 [1], [2]. For each physical path, connecting 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 delay faults in the circuit. A circuit is robustly *delay testable* if and only if every path-delay fault has a robust test. For some Boolean functions, there may not exist any two-level realization which is robustly delay testable. To synthesize for delay fault testability, several approaches [3]–[10] were proposed, such as constrained two-level minimization procedure [3], [4], and use of control variables [5]–[7]. It has been shown that, two-level realizations of most of the symmetric functions are not fully robust testable even if they are prime and irredundant [9]. Transformation techniques are needed to make them delay testable using three- or four-level circuits [7]. Symmetric Boolean functions often appear in logic design [11], and are widely used in cryptology [13], [14].

Manuscript received February 17, 1999; revised January 17, 2000. This work was supported in part by Motorola India Electronics Ltd., Bangalore, India. This paper was recommended by Associate Editor S. Reddy.

S. Chakrabarti is with the Department of Computer Science, University of Kalyani, WB 741 235, India (e-mail: susanta@klyuniv.ernet.in).

S. Das is with the ACM Unit, Indian Statistical Institute, Calcutta 700035, India (e-mail: sandipdas@isical.ac.in).

D. K. Das is with the CSE Department, Jadavpur University, Calcutta 700032, India (e-mail: debeshd@hotmail.com).

B. B. Bhattacharya is with the ACM Unit, Indian Statistical Institute, Calcutta 700035, India (e-mail: bhargab@isical.ac.in).

Publisher Item Identifier S 0278-0070(00)07479-0.

In this paper, we address the synthesis problem of totally symmetric functions for *robust* path-delay fault testability using unate decomposition followed by a factorization technique. The proposed synthesis method guarantees 100% path-delay fault testability, and can be completed in linear time. It leads to a significant reduction of circuit cost in terms of amount of logic and path count as compared to earlier techniques [4], [9].

II. Preliminaries

Let $f(x_1, x_2, \ldots, x_n)$ denote a switching function of n Boolean variables. A *vertex (minterm)* is a product of literals in which every variable appears once. The weight w of a vertex v is defined as the number of uncomplemented literals that appear in v. Let C be a set of prime implicants that covers all 1-vertices (true minterms) of f. The cover C is said to be minimal, if no subset of C is also a cover of f. The logical sum of the elements in C is a sum-of-products (s-o-p) expression of f. The cover C is said to be *minimum*, if there exists no other cover of f with fewer prime implicants than C. Let I be an implicant in a cover C, and v, a vertex in P. A vertex v is said to be a *relatively essential vertex* of P, if there exists no other implicant in C that also covers v. A switching function $f(x_1, x_2, \ldots, x_n)$ is said to be *totally symmetric* with respect to the literals $(x_1, x_2, \ldots, x_n)$ if it is invariant under any permutation of the literals [11]. Total symmetry can be expressed in terms of a set of integers (called a-numbers) $A = \{a_1, \ldots, a_j, \ldots a_k\}$, where $A \subseteq \{0, 1, 2, \ldots, n\}$; all the vertices with weight $w \in A$ will appear as true minterms in the function. An n-variable symmetric function is denoted as $S^n(a_i, \ldots, a_j, \ldots, a_k)$. For n variables, we can construct $2^{n+1} - 2$ totally symmetric functions (excluding constant functions 0 and 1). A symmetric function is said to be *consecutive*, if the set A consists of only consecutive integers $\{a_i, a_{i+1}, \ldots, a_r\}$, and is denoted by $S^n(a_i\text{-}a_r)$.

Each prime implicant of a consecutive symmetric function $S^n(a_i\text{-}a_r)$ consists of exactly $(n - a_r + a_i)$ literals, out of which a_i literals are in the true form, and $(n - a_r)$ literals are in the complemented form. Further, the minimum cover of $S^n(a_i\text{-}a_r)$ consists of m prime implicants, where $m = \max\{\binom{n}{a_i}, \binom{n}{a_r}\}$ [12].

A totally symmetric function $S^n(A)$ can always be expressed uniquely as a union of maximum 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 $i \neq j$ [12].

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

III. Synthesis of Consecutive Symmetric Functions to Achieve Robust Testability

It has been shown [9] that a two-level irredundant circuit realizing a minimum cover of a consecutive symmetric function $S^n(a_i\text{-}a_r)$, is robustly testable for all path-delay faults if and only if 1) $a_i = 0$, or $a_r = n$, or 2) $\binom{n}{a_i} = \binom{n}{a_r}$. If the above condition is not satisfied, the number of untestable paths in the circuit becomes at least $[\{\binom{n}{a_i} \sim \binom{n}{a_r}\} + 1] \cdot \min(a_i, n - a_r)$. Complete robust testability can be achieved by designing a three- or four-level circuit obtained via

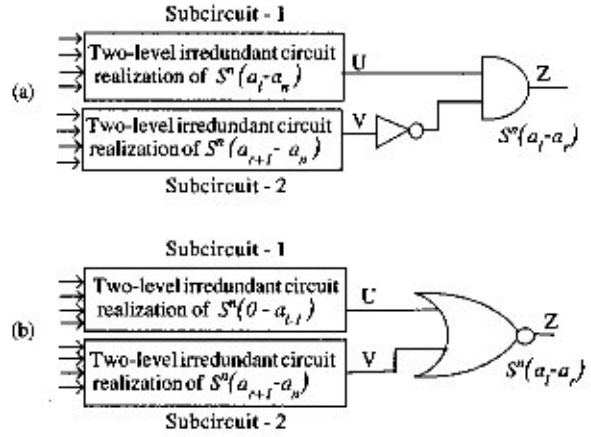

Fig. 1. Robustly testable circuit for Case 1) and Case 2).

factorization. We here propose a new and cost-effective technique for synthesis.

The proposed method of synthesizing $S^n(a_l$-$a_r)$, for full (100%) robust path-delay fault testability is based on the following three key observations:

1) $S^n(a_l$-$a_r)$ is unate if $a_l = 0$, or $a_r = n$;
2) a two-level irredundant AND-OR realization of a unate symmetric function is fully robust delay testable. If $a_l = a_r$, then each minterm will be a prime implicant by itself, and a minimum two-level realization will be 100% robust testable since $\binom{n}{a_l} = \binom{n}{a_r}$;
3) every consecutive symmetric function $S^n(a_l$-$a_r), a_l \neq a_r$ where $a_l \neq 0, a_r \neq n$, can be expressed as a logical composition (e.g., AND, NOR) of two consecutive symmetric functions which are unate, and the resulting composite circuit realizing $S^n(a_l$-$a_r)$ is 100% robustly delay fault testable.

A. Synthesis Technique

The following theorem captures the proposed composition technique.

Theorem 1: $S^n(a_l$-$a_r), a_l \neq a_r, l < r$, can be expressed as a composition of two unate and consecutive symmetric functions as follows:

1) $S^n(a_l$-$a_r) = S^n(a_l$-$a_n) \cdot \overline{S^n(a_{r+1}$-$a_n)}$
2) $S^n(a_l$-$a_r) = \overline{S^n(0$-$a_{l-1})} + S^n(a_{r+1}$-$a_n)$
3) $S^n(a_l$-$a_r) = S^n(0$-$a_r) \cdot S^n(0$-$a_{l-1})$
4) $S^n(a_l$-$a_r) = S^n(0$-$a_r) \cdot S^n(a_l$-$a_n)$

Proof: For Case 1), clearly, the functions $S^n(a_l$-$a_n)$ and $S^n(a_{r+1}$-$a_n)$ are unate. Now, the a-numbers of these two functions are $(l, l-1, \ldots, n)$ and $(r+1, \ldots, n)$ respectively. Therefore for the right-hand side, the a-numbers of the composite function

$$\begin{aligned} &= (l, l+1, \ldots, n) \cap \overline{(r+1, \ldots, n)} \\ &= (l, l+1, \ldots, n) \cap (0, 1, \ldots, l, l+1, \ldots, r) \\ &- (l, l+1, \ldots, r) \\ &= a\text{-numbers of } S^n(a_l\text{-}a_r). \end{aligned}$$

For cases 2)–4), similar arguments will apply. ■

The schemes of the circuit for cases 1) and 2) are shown in Fig. 1(a) and (b) respectively.

Example 2: An irredundant two-level realization of $S^6(4,5)$ is not robustly delay testable. In our proposed technique of synthesis-for-testability, it is implemented as per Case 1),

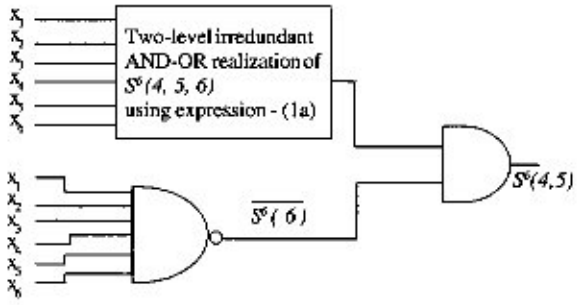

Fig. 2. Robustly testable circuit for $S^6(4,5)$.

$S^6(4,5) = S^6(4,5,6)\overline{S^6(6)}$. The minimum s-o-p expression of $S^6(4,5,6)$ and $S^6(6)$ are given by

$$\begin{aligned} S^6(4,5,6) = {} & x_1x_2x_4x_5 + x_1x_3x_4x_5 + x_1x_2x_4x_6 \\ & + x_1x_3x_4x_6 + x_1x_2x_5x_6 + x_1x_3x_5x_6 \\ & + x_2x_3x_4x_5 + x_2x_3x_4x_6 + x_2x_3x_5x_6 \\ & + x_1x_2x_5x_6 + x_1x_4x_5x_6 + x_2x_4x_5x_6 \\ & + x_3x_4x_5x_6 + x_1x_2x_3x_6 + x_2x_3x_5x_6 \end{aligned} \quad (1a)$$

and

$$S^6(6) = x_1x_2x_3x_4x_5x_6.$$

The testable circuit for $S^6(4,5)$ is shown in Fig. 2.

Theorem 2: Any consecutive symmetric function $S^n(a_l$-$a_r), (a_l \neq a_r)$, implemented as above, is robustly path-delay fault testable.

Proof: Case 1): Let $v(a_i)$ represent a vertex with weight a_i. Consider the circuit of $S^n(a_l$-$a_r)$ as in Fig. 1 $(a_l \neq a_r)$. The functions $S^n(a_l$-$a_n)$ and $S^n(a_{r+1}$-$a_n)$ are unate. Both have only uncomplemented variables in their minimum covers. Therefore, each vertex $v(a_l)$ $[v(a_{r-1})]$ is a relatively essential vertex in $S^n(a_l$-$a_n)\{S^n(a_{r+1}$-$a_n)\}$. Conversely, in each prime implicant of $S^n(a_l$-$a_n)\{S^n(a_{r+1}$-$a_n)\}$, there exists a vertex $v(a_l)\{v(a_{r-1})\}$, which is essential.

Therefore, the subcircuit 1 (subcircuit 2) is robustly path-delay fault testable for all paths enumerating from the inputs up to the line $U(V)$. For rising [respectively, falling] transitions in the subcircuit 1, the test set consists of all ordered pairs, $\{v(a_{l-1}), v(a_l)\}$ [respectively, $\{v(a_l), v(a_{l-1})\}$] that lie at unit Hamming distance. Similarly, for subcircuit 2, the test pairs are of the form $\{v(a_r), v(a_{r+1})\}$ for rising transitions, and $\{v(a_{r+1}), v(a_r)\}$ for falling transitions.

We can now prove that the overall circuit is robustly testable. Since, $a_l \neq a_r$, for all the vectors of weight a_{l-1} and a_l, the logic value at V is zero and, hence, the line U is set to 1 which is the noncontrolling value for the AND gate. Hence, all paths through the subcircuit 1 are robustly testable at the circuit output Z. On the other hand, for all vectors of weight a_r or a_{r+1}, the responses of subcircuit 1 at line U will be 1, which is again the noncontrolling value for the following AND gate. Thus, all paths passing through V are also robustly testable at the output Z.

Case 2): Consider implementation of $S^n(a_l$-$a_r)$ using two unate functions $S^n(0$-$a_{l-1})$ and $S^n(a_{r+1}$-$a_n)$ as in Fig. 1(b). The former (latter) is unate with respect to all complemented (uncomplemented) literals, and by similar argument, each prime implicant has a relatively essential vertex of weight a_{l-1} (a_{r+1}). Thus, the subcircuit 1 (subcircuit 2) is robustly delay fault testable for all paths from the inputs up to the line $U(V)$. Since $l < r$, the response of $S^n(a_{r-1}$-$a_n)$ at V is 0 for all the vectors with weight a_{l-1} or a_l. In other words, for all test pairs, $\{v(a_{l-1}), v(a_l)\}$ and $\{v(a_l), v(a_{l-1})\}$, the logic value at V is 0.

TABLE I
SYNTHESIS COST

Case	Circuit cost	Test cost	Useful when
i)	$\binom{n}{a_{r+1}}(a_{r+1}+1)+$ $\binom{n}{a_l}(a_l+1)+2$	$\binom{n}{a_{r+1}}(a_{r+1})+$ $\binom{n}{a_l}a_l$	$a_l > \lfloor n/2 \rfloor$ $a_r > \lfloor n/2 \rfloor$
ii)	$\binom{n}{a_{l-1}}(n-a_{l-1}+1)$ $+\binom{n}{a_{r+1}}(a_{r+1}+1)+2$	$\binom{n}{a_{l-1}}(n-a_{l-1})$ $+\binom{n}{a_{r+1}}a_{r+1}$	$a_l \leq \lfloor n/2 \rfloor$ $a_r \geq \lceil n/2 \rceil$
iii)	$\binom{n}{a_{l-1}}(n-a_{l-1}+1)$ $+\binom{n}{a_r}(n-a_r+1)+2$	$\binom{n}{a_{l-1}}(n-a_{l-1})$ $+\binom{n}{a_r}(n-a_r)$	$a_l < \lfloor n/2 \rfloor$ $a_r < \lfloor n/2 \rfloor$

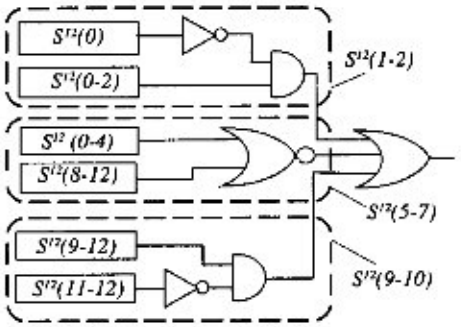

Fig. 3. Testable circuit realizing $S^{12}(1,2,5,6,7,9,10)$.

Therefore, all transitions along any path through the subcircuit 1, will reach at the circuit output Z, without any interference from subcircuit 2, as the line V is set to the non controlling value of the NOR gate during testing of subcircuit 1. Similarly, all paths through the subcircuit 2 can be robustly tested by applying suitable test pairs $\{v(a_r), v(a_{r+1})\}$ and $\{v(a_{r+1}), v(a_r)\}$, as for all such vertices, the line U is set to 0.

d*Case 3) and Case 4):* Similar to Case 1). ■

Remark: The complete set of two-pattern tests for testing path-delay faults in the proposed design can be determined immediately, without running an automatic test pattern generation (ATPG).

B. Circuit Cost

The suitability of the above synthesis methods based on cases 1)–3) as mentioned in Theorem 1, and the corresponding circuit cost (the number of gate inputs) and test cost (the number of paths) of the composite circuit realizing $S^n(a_l,a_r)$, are summarized in Table I. Case 4) does not yield any cost-effective solution.

To synthesize $S^n(a_l,a_r)$ with minimum cost, we thus need an appropriate pair of unate symmetric functions among $S^n(0,a_{l-1})$, $S^n(0,a_r)$, $S^n(a_l,a_n)$, and $S^n(a_{r-1},a_n)$, depending on the relative values of a_l, a_r, and n. The resulting circuit has at most 3 logic levels.

IV. SYNTHESIS OF NONCONSECUTIVE SYMMETRIC FUNCTIONS

To synthesize a nonconsecutive symmetric function for 100% robust path-delay fault testability, it is first expressed as a union of several maximum consecutive symmetric functions, and then testable realization for each of the constituent consecutive symmetric functions is derived via unate decomposition, and finally they are OR-ed together. Thus, the overall circuit requires at most four logic levels. It is now easy to prove that the overall circuit becomes robustly testable.

Example 3: The testable circuit realizing the nonconsecutive symmetric function $S^{12}(1,2,5,6,7,9,10)$ of Example 1, is shown in Fig. 3, where each of the unate components within the boxes is synthesized as a two-level irredundant circuit.

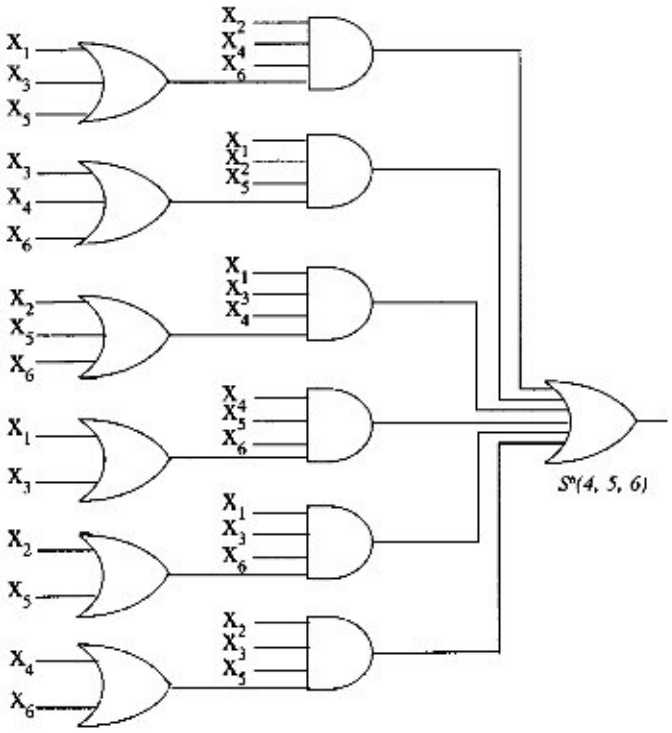

Fig. 4. Optimum factorization of $S^6(4,5,6)$.

Theorem 3: Every nonconsecutive symmetric function $S^n(A)$ synthesized as above, is robustly testable for all path-delay faults.

Proof: Since A is partitioned into maximal consecutive sets of integers, for every pair consecutive symmetric functions $S^n(A_i)$ and $S^n(A_j), 1 \leq i,j \leq m$, corresponding to the partition of A, the integers in $A_i \cup A_j$ cannot be consecutive. There must a discontinuity between the sequences of A_i and A_j. Therefore, for each test pair of $S^n(A_i)$, the output of $S^n(A_j) = 0$, for all $j \neq i$. Hence, each consecutive component can be tested independently at the primary output of the nonconsecutive symmetric function. Hence the proof. ■

V. REDUCTION OF LOGIC BY ALGEBRAIC FACTORIZATION

The circuit cost of the proposed design can further be reduced by factoring out common literals from the product terms at the cost of increasing the number of logic levels by one or more. However, we will restrict to only one extra level, otherwise the circuit delay may increase significantly. Optimal factorization for minimum circuit cost, is a difficult combinatorial problem for an arbitrary symmetric function, although a simple technique can be used for consecutive symmetric functions that are unate.

For example, the expression-(1a) of $S^6(4,5,6)$, can be factored algebraically as

$$\begin{aligned} S^6(4,5,6) &= x_2x_4x_5(x_1+x_3+x_6)+x_1x_2x_5(x_4+x_3+x_6)\\ &\quad + x_1x_3x_4(x_5+x_2+x_6)+x_1x_3x_5(x_4+x_3)\\ &\quad + x_1x_3x_6(x_4+x_5)+x_2x_3x_5(x_1+x_6). \end{aligned} \tag{1b}$$

Realization of the above factored expression by AND-OR gates requires 45 gate inputs and 3 levels of logic (Fig. 4).

Optimal algebraic factorization that leads to minimum number of gate inputs in three-level synthesis can be achieved by minimizing the number of factors.

A. Formulation

A nonconstant unate consecutive symmetric function is always of the form: i) $S^n(h\text{-}n)$, $0 < h \leq n$, which consists of all uncomplemented literals in its s-o-p form; ii) $S^n(0\text{-}p)$, $0 \leq p < n$, which consists of all complemented literals in its s-o-p form.

The case for which $h = 0$ or $p = n$, leads to a constant function and, hence, can be ignored.

To illustrate the technique, we consider Case 1) only.

It may be noted that $S^n(h\text{-}n) = \overline{S^n\{0\text{-}(h-1)\}}$. Therefore, the same function can be synthesized by complementing the output of $S^n\{0\text{-}(h-1)\}$, i.e., by using an AND-NOR structure. For $h < \lfloor n/2 \rfloor$, it can be shown that circuit-cost$[S^n(h\text{-}n)] >$ circuit-cost$[S^n\{0\text{-}(h-1)\}]$. Thus, we need to consider factorization of $S^n(h\text{-}n)$ only for the case where $\lfloor n/2 \rfloor \leq h \leq n$.

Our goal is to determine a suitable factorization that leads an optimal three-level logic synthesis for unate and consecutive symmetric functions. Thus, each factored term should be of the form: $x_{i_1}x_{i_2}\ldots x_{i_{q-1}}\{x_{j_1}+x_{j_2}+\cdots+x_{j_g}\}$, $q \geq 1$, where x_i's and x_j's are distinct and belong to the set of variables $X = \{x_1, x_2, \ldots, x_n\}$.

The function $S^n(h\text{-}n)$ is unate and consecutive, having $\binom{n}{h}$ prime implicants, and h literals in each prime implicant. Therefore, the above factorization problem is equivalent to the following combinatorial cover problem: select a minimum set of $(h-1)$-out-of-n combinations that cover all $\binom{n}{h}$ h-out-of-n combinations. A $(h-1)$-out-of-n combination is said to *cover* a h-out-of-n combination, if the $(h-1)$ set of literals in the former is a subset of the h-set of literals of the latter. Since the general cover problem is NP-hard, we present a heuristic technique of solving this problem using combinatorial grouping techniques. Empirical evidence shows that the proposed heuristic works well for all $0 < h \leq n$, but produces solutions very close to the optimum, when $\lfloor n/2 \rfloor \leq h \leq n$.

B. Algorithm

Input: S-o-p expression E of a unate consecutive symmetric function $S^n(h\text{-}n)$;/* Given the set P of prime implicants, and the variable set $X = \{x_1, x_2, \ldots, x_n\}$. */

Output: Algebraically factored expression E' in which each term is of the form: $x_{i_1}x_{i_2}\ldots x_{i_{k-1}}\{x_{j_1}-x_{j_2}+\cdots+x_{j_g}\}$, such that $E' = E$; /* E' can be directly synthesized using a three-level OR-AND-OR logic. */

```
{
Initialization: i ← 1, E' ← ∅;
repeat
    Select x_i, x_{i+1} ∈ X;
    Partition the set of prime implicants into three groups;
    Group 1: collect prime implicants that contain
        either x_i, x_{i+1} but not both;
        factor suitable pairs of them as
            x_{i_1}, ..., x_{i_{k-1}}{x_i + x_{i+1}},
        and form a logical sum T_i of all such factors;
    Group 2: collect prime implicants that contain
        neither x_i nor x_{i+1};
        logically add each of them to T_i, factoring
        it with a matching term, and set E' ← E' − T_i;
    Group 3: P ← P\{Group1 ∪ Group2};
        i ← i + 2;
        X ← X\{x_i, x_{i+1}};
until P is empty or contains a single term;
if P contains a single term, logically add it to E';
}
```

Example 4: Consider the s-o-p of $S^6(4,5,6)$ as in exp. (1a). Choose two variables (say x_1, x_2). Partition the set of prime implicants into three disjoint groups as follows:

Group 1: $x_1x_3x_4x_5$, $x_1x_3x_4x_6$, $x_1x_3x_5x_6$, $x_1x_4x_5x_6$, $x_2x_3x_4x_5$, $x_2x_3x_4x_6$, $x_2x_3x_5x_6$, $x_2x_4x_5x_6$;

Group 2: $x_3x_4x_5x_6$;

Group 3: $x_1x_2x_3x_5$, $x_1x_2x_4x_5$, $x_1x_2x_3x_6$, $x_1x_2x_4x_6$, $x_1x_2x_5x_6$, $x_1x_2x_5x_6$.

The terms in Group 1 can be factored as: $x_3x_4x_5(x_1+x_2)$, $x_3x_4x_6(x_1+x_2)$, $x_3x_5x_6(x_1+x_2)$, $x_4x_5x_6(x_1+x_2)$.

Each term in Group 2 must have a common factor with at least one term in Group 1, with which it can be factored. For example, the term $x_3x_4x_5x_6$ belonging to Group 2, can be merged with Group 1 as: $x_3x_4x_5(x_1+x_2+x_6)$, $x_3x_4x_6(x_1+x_2)$, $x_3x_5x_6(x_1+x_2)$, $x_4x_5x_6(x_1+x_2)$.

Thus, the remaining prime implicants of Group 3 are devoid for x_1 and x_2. The same process is now repeated by choosing another pair of literals from $X\backslash\{x_1, x_2\}$, till all prime implicants are factored, or at most a single prime implicant is left.

In the second pass, choosing x_3 and x_4, we get: $x_1x_2x_3(x_3-x_4+x_6)$, $x_1x_2x_6(x_3+x_4)$. Combining them, we get: $x_3x_4x_5(x_1+x_2+x_6)$, $x_3x_4x_6(x_1+x_2)$, $x_3x_5x_6(x_1+x_2)$, $x_4x_5x_6(x_1+x_2)$, $x_1x_2x_5(x_3+x_4+x_6)$, $x_1x_2x_6(x_3+x_4)$.

At this stage, we are left with only one prime implicant which is $x_1x_2x_3x_4$, and the process stops.

The final factored form can be obtained as

$$\begin{aligned} S^6(4,5,6) = {} & x_3x_4x_5(x_1-x_2+x_6)+x_3x_4x_5(x_1-x_2) \\ & + x_3x_5x_6(x_1-x_2)+x_4x_5x_6(x_1+x_2) \\ & + x_1x_2x_5(x_1-x_4+x_6)+x_1x_2x_6(x_1-x_4) \\ & + x_1x_2x_3x_4. \end{aligned} \tag{1c}$$

.

The number of factors in (1c) is seven, and the number of gate inputs in its corresponding three-level realization is 49.

Theorem 4: The above algorithm generates algebraically factored expression E', such that 1) $E = E'$, 2) E' can be directly realized using a three-level OR-AND-OR logic, and 3) the number of factors in E' is

$$\begin{aligned} &= \left[\binom{n-2}{h-1}+\binom{n-4}{h-3} + \binom{n-6}{h-5}\cdots\binom{n-h-1}{2}+1\right]. \quad \text{if } h \text{ is odd;} \\ &= \left[\binom{n-2}{h-1}+\binom{n-4}{h-3} + \binom{n-6}{h-5}\cdots\binom{n-h}{1}+1\right]. \quad \text{if } h \text{ is even.} \end{aligned}$$

Proof: Parts 1) and 2) follow immediately from the theme of the algorithm. To prove the counting result, we proceed as follows: In Group 1, the number of terms containing either x_i or x_{i+1}, but not both $= 2\cdot\binom{n-2}{h-1}$. So, the number of factors after the first iteration $= \binom{n-2}{h-1}$. In Group 2, there are $\binom{n-2}{h}$ terms, and each of them must have a common factor with some of those in Group 1. The same process is now repeated by choosing another pair from $X\backslash\{x_i, x_{i+1}\}$, till all prime implicants are factored. Counting this way, we get, the number of factors $= \binom{n-2}{h-1}+\binom{n-4}{h-3}+\ldots$.

However, depending on whether h is odd or even, the terminating terms become different, and the theorem follows. ■

Corollary 1: The number of gate inputs and paths in a three-level synthesis of E' are given by

i) the number of gate inputs

$$= \left[\binom{n-2}{h-1} + \binom{n-4}{h-3} + \binom{n-6}{h-5} \cdots \binom{n-h+1}{2} + 1\right] \cdot (h+1) + \binom{n}{h}, \quad \text{if } h \text{ is odd:}$$

$$= \left[\binom{n-2}{h-1} + \binom{n-4}{h-3} + \binom{n-6}{h-5} \cdots \binom{n-h}{1} + 1\right] \cdot (h+1) + \binom{n}{h} - 1, \quad \text{if } h \text{ is even}$$

ii) the number of paths

$$= \left[\binom{n-2}{h-1} + \binom{n-4}{h-3} + \binom{n-6}{h-5} \cdots \binom{n-h+1}{2} + 1\right] \cdot (h-1) + \binom{n}{h}, \quad \text{if } h \text{ is odd:}$$

$$= \left[\binom{n-2}{h-1} + \binom{n-4}{h-3} + \binom{n-6}{h-5} \cdots \binom{n-h}{1} + 1\right] \cdot (h-1) + \binom{n}{h} - 1, \quad \text{if } h \text{ is even.}$$

C. Complexity Analysis

The proposed synthesis technique consists of three steps: 1) expressing the given symmetric function as a logical sum of maximum consecutive symmetric functions, 2) synthesizing each of them via unate decomposition, and 3) factoring algebraically each of the unate components by the above heuristic. The minimum covers of unate symmetric functions can be determined readily. Thus, the overall synthesis process including the heuristic factorization algorithm terminates in time linear in the size of the cover. Since the factorization is algebraic, it preserves robust testability [3].

D. Experimental Results

Corollary 1 can be used to calculate the number of gate inputs and paths in the synthesized circuits according to the proposed method.

We also compute the circuit and test cost for synthesizing $S^n(a_l\text{-}a_r)$, following the method of Ke and Menon [9]. It can be shown [15] that: the number of gate inputs $= \alpha(\beta-\gamma+1)+(\beta+1)(\delta-1)+\gamma+2$; and the number of paths $= \alpha(\beta-\gamma)+\beta(\delta-1)+\gamma$, where, $\alpha = \{\binom{n}{a_l} \sim \binom{n}{a_r}\}+1, \beta = (n-a_r+a_l), \gamma = \min(a_l, n-a_r), \delta = \min\{\binom{n}{a_l}, \binom{n}{a_r}\}$.

Table II presents a comparative study of circuit cost (the number of gate inputs) and test cost (the number of paths) for various examples of consecutive symmetric functions. It can be readily seen that our approach produces a significant amount of savings compared to the earlier method of Ke and Menon [9].

TABLE II
COMPARISON OF CIRCUIT AND TEST COST FOR CONSECUTIVE SYMMETRIC FUNCTIONS.

Functions $S^n(a_l\text{-}a_r)$	Number of gate inputs			Number of paths		
	As in [9]	Our Method	Savings (%)	As in [9]	Our Method	Savings (%)
$S^5(3,4)$	47	33	30	35	23	34
$S^5(1,2)$	47	33	30	35	23	34
$S^6(4,5)$	83	57	31	66	41	37
$S^5(1,2)$	83	57	31	66	41	37
$S^6(3,4)$	112	75	33	90	53	41
$S^6(2,3)$	112	75	33	90	53	41
$S^7(4,5)$	219	140	36	182	102	43
$S^7(1\text{-}5)$	72	43	40	49	33	32
$S^7(1,2)$	135	84	38	112	64	42
$S^7(3,4)$	245	152	38	210	114	45
$S^7(2,3)$	219	140	36	182	102	43
$S^8(5,6)$	394	230	42	336	176	47
$S^8(2\text{-}6)$	140	82	41	112	64	42
$S^8(2,3)$	394	230	42	336	176	47
$S^8(4,5)$	520	308	41	448	236	47
$S^8(3,4)$	520	308	41	448	236	47
$S^9(5,6)$	1010	568	44	882	442	49
$S^9(2\text{-}6)$	410	219	47	324	175	45
$S^9(2,3)$	662	383	42	576	301	47
$S^9(4,5)$	1134	658	42	1008	516	48
$S^9(3,4)$	1010	568	44	882	442	49
$S^{10}(6,7)$	1832	1011	45	1620	805	50
$S^{10}(3\text{-}7)$	840	468	44	720	382	46
$S^{10}(3,4)$	1832	1011	45	1620	805	50
$S^{10}(5,6)$	2354	1298	45	2100	1034	50
$S^{10}(4,5)$	2354	1298	45	2100	1034	50
$S^{11}(6,7)$	4556	2435	46	4092	1957	52
$S^{11}(3\text{-}7)$	2147	1110	48	1815	916	49
$S^{11}(3,4)$	3137	1677	46	2805	1365	51
$S^{11}(5,6)$	5082	2742	46	4620	2220	51
$S^{11}(4,5)$	4556	2435	46	4092	1957	52
$S^{12}(7,8)$	8318	4332	48	7524	3548	52
$S^{12}(4,5)$	8318	4332	48	7524	3548	52
$S^{12}(6,7)$	10430	5465	48	9504	4469	52
$S^{12}(5,6)$	10430	5465	48	9504	4469	52
$S^{13}(7,8)$	20165	10263	49	18447	8451	54
$S^{13}(4\text{-}8)$	10584	5338	49	9295	4494	51
$S^{13}(4,5)$	14445	7432	48	13156	6184	52
$S^{13}(6,7)$	22308	11520	48	20592	9530	53
$S^{13}(5,6)$	20165	10263	49	18447	8451	54
$S^{14}(8,9)$	37039	18598	50	34034	15540	54
$S^{14}(5\text{-}9)$	22022	11264	49	20020	9578	52
$S^{14}(5,6)$	37039	18598	50	34034	15540	54
$S^{14}(7,8)$	45476	22879	50	42042	19081	54
$S^{14}(6,7)$	45476	22879	50	42042	19081	54
$S^{15}(8,9)$	87947	43200	51	81510	36234	55
$S^{15}(5\text{-}9)$	50052	24673	51	45045	21085	53
$S^{15}(5,6)$	65067	32314	50	60060	27356	54
$S^{15}(7,8)$	96525	47952	50	90090	40362	55

Table II presents a comparative study of circuit cost (the number of gate inputs) and test cost (the number of paths) for various examples of consecutive symmetric functions. It can be readily seen that our approach produces a significant amount of savings compared to the earlier method of Ke and Menon [9].

VI. CONCLUSION

This paper introduces a new concept of unate decomposition which is shown to be useful for synthesis of symmetric functions targeted to achieve path-delay fault testability. Synthesis of symmetric functions is an important issue, as most of these functions do not have delay testable two-level implementations. The proposed method guarantees complete robust path-delay fault testability, and the design can be completed in linear time. A novel factorization technique is also suggested that preserves testability, and yields very low circuit cost compared to earlier approaches, at the cost of increasing the number of logic levels by at most one. As the number of paths decreases significantly, the total testing time is drastically reduced. Further, all the test pairs can be determined readily from the corresponding factored forms without any need of running an ATPG.

ACKNOWLEDGMENT

The authors wish to thank S. Maitra of the Indian Statistical Institute, Calcutta, for his constructive comments.

REFERENCES

[1] G. L. Smith, "Model for delay faults based upon paths," in *Proc. Int. Test Conf.*, 1985, pp. 342–349.

[2] C. J. Lin and S. M. Reddy, "On delay fault testing in logic circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 694–703, Sept. 1987.

[3] S. Devadas and K. Keutzer, "Synthesis of robust delay-fault-testable circuits: Theory," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 87–101, Jan. 1992.
[4] ——, "Synthesis of robust delay-fault-testable circuits: Practice," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 227–300, Mar. 1992.
[5] I. Pomeranz and S. M. Reddy, "Achieving complete delay fault testability by extra inputs," in *Proc. Int. Test Conf.*, 1991, pp. 273–282.
[6] V. A. Vardanian, "On completely robust path delay fault testable realization of logic functions," in *Proc. VLSI Test Symp.*, 1996, pp. 302–307.
[7] N. K. Jha, I. Pomeranz, S. M. Reddy, and R. J. Miller, "Synthesis of multi-level combinational circuits for complete robust path delay fault testability," in *Proc. FTCS*, 1992, pp. 280–287.
[8] A. K. Pramanick and S. M. Reddy, "On the design of path delay fault testable combinational circuits," in *Proc. FTCS*, 1990, pp. 374–381.
[9] W. Ke and P. R. Menon, "Delay-testable implementations of symmetric functions," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 772–775, June 1995.
[10] I. Pomeranz and S. M. Reddy, "On synthesis-for-testability of combinational logic circuits," in *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 126–132.
[11] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw Hill, 1977.
[12] D. L. Dietmeyer, "Generating minimal covers of symmetric function," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 710–713, May 1993.
[13] C. Ding, G. Xiao, and W. Shan, *The Stability Theory of Stream Ciphers*. Berlin, Germany: Springer-Verlag, 1991. LNCS.
[14] Y. X. Yang and B. Guo, "Further enumerating Boolean functions of cryptographic significance," *J. Cryptol.*, vol. 8, no. 3, pp. 115–122, 1995.
[15] S. Chakrabarti, "Studies in redundancy and testable design of logic circuits," Ph.D. thesis, Univ. Calcutta, Calcutta, India, 1998.

New Multivalued Functional Decomposition Algorithms Based on MDDs

Craig M. Files and Marek A. Perkowski

***Abstract*—This paper presents two new functional decomposition partitioning algorithms that use multivalued decision diagrams (MDDs). MDDs are an exceptionally good representation for generalized decomposition because they are canonical and they can represent very large functions. Algorithms developed in this paper are for Boolean/multivalued input and output, completely/incompletely specified functions with application to logic synthesis, machine learning, data mining and knowledge discovery in databases. We compare the run-times and decision diagram sizes of our algorithms to existing decomposition partitioning algorithms based on decision diagrams. The comparisons show that our algorithms are faster and do not result in exponential diagram sizes when decomposing functions with small bound sets.**

***Index Terms*—Algorithms, logic design, unsupervised learning.**

I. Introduction

Functional decomposition is known as expressing a function as a composition of two or more functions. While many papers were written about the topic of functional decomposition there was no comprehensive approach until Ashenhurst presented a unified theory of functional decomposition, and for the first time defined its basic properties in [1], [2]. Curtis used the theorems of Ashenhurst to develop a generalized form of decomposition in [3] and [4]. There have been many other proposed types of functional decompositions since the advent of Curtis decomposition. But, the fundamentals of Ashenhurst and Curtis decomposition provide essential insight into a wide range of functional decomposition types.

Two new functional decomposition partitioning algorithms that are based on multivalued decision diagrams (MDDs) [5] are presented in this paper: PARTITION and EVAL. Both algorithms are compared to the existing *cut_level* and LPV (Lai, Pedram, and Vrudhula) binary decision diagram (BDD)-based functional decomposition partitioning algorithms developed by Lai, Pedram, and Vrudhula [6]–[8]. The *cut_level* algorithm is very well known, but is based on reordering the variables in the decision diagram. This can be a problem because variable reordering may lead to decision diagrams of exponential size [9]. This is the advantage of the PARTITION algorithm over the *cut_level* algorithm because the PARTITION algorithm does not reorder variables in the decision diagram. The LPV algorithm is much faster than the *cut_level* algorithm and can quickly evaluate many partitions, but the LPV algorithm can only be used to determine *column multiplicities*, to decompose a function the *cut_level* algorithm must be used.

The advantage of the EVAL algorithm over the LPV algorithm is based on the way the two algorithms construct partition tables and determine column equalities. The LPV algorithm constructs a partition table by constructing each column in the table. After the partition table is created each pair of columns is checked for equality. For completely and incompletely specified functions, two columns are equal if their encoded integer values are equal. An extension to the LPV algorithm for incompletely specified functions is presented in this paper to find columns that are compatible (by setting output don't cares in the columns, the two columns can be made equal).

The EVAL algorithm constructs the partition table by rows and checks if two columns are equal while constructing the partition table. This makes it possible to determine that two columns are not equal or not compatible before completing their construction. Of course, if the two columns are equal then the two columns must be fully constructed. The advantage of the EVAL algorithm is that after the construction of the partition table no extra computation is needed. The LPV algorithm must construct the partition table and then determine column equalities and column compatibilities.

Section II gives the general notations for functional decomposition. Section III presents the Lai, Pedram, and Vrudhula algorithms and our new MDD-based algorithms. Section IV shows the experimental results of the algorithms and the paper is concluded in Section V.

II. General Decomposition

The decomposition of a function can be an expression of the function in terms of a composition of other functions. For example, if $f(x_0, x_1, x_2, x_3) = F(\Phi(x_0, x_1), x_2, x_3)$, then the term on the right is a decomposed function that is equivalent in behavior to the original function f.

In general, an n-input, single output Boolean function, $f: \{0,1\}^n \rightarrow \{0,1\}$, has the set of input variables $X = \{x_0, x_1, \ldots, x_{n-1}\}$. The number of variables in set X is denoted by $|X|$.

Definition 1: Let $A \subset X$ and $B \subset X$, where $A \neq \emptyset$ and $B \neq \emptyset$. A **partition**, denoted as $A|B$, exists if $A \cap B = \emptyset$ and $A \cup B = X$.

Definition 2: A function $f(x_0, x_1, \ldots, x_{n-1})$ has an **Ashenhurst simple disjunctive decomposition** [1], if f can be represented by $f = F(\Phi(B), A)$. This is known as partitioning the input variables into the **bound set** B and the **free set** A. The variables in B and A are known as **bound variables** and **free variables**, respectively.