VLSI Design
1994, Vol. 2, No. 3, pp. 185–198
Reprints available directly from the publisher
Photocopying permitted by license only

Partitioning Techniques for Built-In Self-Test Design

CHIEN-IN HENRY CHEN
Department of Electrical Engineering, Wright State University, Dayton, OH 45435

(Received November 18, 1989; Revised July 10, 1990)

An efficient, unified algorithm, *Advanced Two-Phase Cluster Partitioning,* is proposed for automated synthesis of pseudo-exhaustive test generator for Built-In Self-Test (BIST) design. A prototype of the algorithm, *Two-Phase Cluster Partitioning,* has been proposed and the hierarchical design procedure is computationally efficient and produces test generation circuitry with low hardware overhead. However, in certain worst case, the algorithm may generate a sub-optimal design which requires more test patterns and/or hardware overhead. In order to generate a globally optimal design, further improvement of two-phase algorithm can be achieved by expanding the design space for the formation of linear sum so that the number of test signals required for pseudo-exhaustive testing can be reduced. We demonstrate the effectiveness of our approach by presenting detailed comparisons of our results against those that would be obtained by existing techniques.

Key Words: *Built-In Self-Test (BIST), Pseudo-exhaustive, Clique partitioning, Maximum Test Concurrency (MTC), Linear sum*

1. INTRODUCTION

Testing a circuit consists of applying an input sequence and observing the resulting output sequence. With respect to test generation, the central problem is that in today's IC technology a highly complex chip has low accessibility of internal circuit nodes (controllability and observability). The weak accessibility makes traditional testing techniques costly and ineffective. Especially, modern RISC processors incorporate a hundred of thousands of transistors to perform the bulk of the task of a complete computer system on a single chip. This level of integration has substantially increased the complexity of testing modern RISC processors to the point where classical functional based testing methods are simply inadequate. Thus, design for testability techniques are in demand.

Built-In Self-Test (BIST) has been proposed as a powerful technique for addressing the highly complex problems of VLSI testing [1–9]. The basic idea is to include the test generator and evaluator into the design and to perform the testing internal to the chip. A widely adopted design approach is one in which BIST is combined with some type of scan design methodology. In this situation, the internal state of the system under test is completely accessible so that the BIST hardware need only create and analyze tests for purely combinational circuits. While exhaustively checking all possible input patterns is neither feasible nor required in many situations. For a general multi-input, multi-output combinational circuit in which no output is a function of all input variables, pseudo-exhaustive testing may be applied. For circuits of this type, it is possible to obtain the same amount of information as obtained in exhaustive testing, but with a reduced number of test patterns. One needs to find sets of input pins which can share common test signals. This problem is completely analogous to resource sharing in data-path synthesis, and certain modifications of efficient algorithms developed for the latter can be applied to the test generation problem.

There have been many variations on the basic idea of BIST design. The first method for pseudo-exhaustive test pattern generation was proposed in [10] in which a syndrome drive counter (SDC) designed by a minimum covering procedure was proposed for generating test patterns. In that technique, it can be checked whether the different inputs of the CUT (circuit under test) can be tested by using the same test signal. But, test time may be still too long when

only a few inputs share the same test signal. Also, no attempt is made to find the minimum number of test signals for the non-MTC circuits. If the number of required test signals is equal to the maximum number of inputs upon which any output depends, the circuit is called an MTC (Maximum Test Concurrency) circuit [11]. Using the proposed technique, fewer test signals are needed for the non-MTC circuits.

Two universal testing techniques have been proposed to reduce the number of test signals for pseudo-exhaustive testing. The first technique, verification testing [11], uses constant weight counters (CWC) to implement pseudo-exhaustive generators. A major problem with this approach is that input stimulus generation must be computed, and that is an NP-complete problem. Therefore, for circuits having a higher order m-out-of-n code [11], CWC is very costly to implement. Next, a combination of LFSRs and exclusive-or gates was proposed in [12–13]. However, the techniques proposed in these papers did not attempt to allocate the minimum number of required test signals, nor did they attempt to minimize the extra hardware overhead (i.e. XOR gates). Because these problems are NP-complete, a universal procedure (LFSRs/XORs) was proposed in [12] to solve the problem where an upper bound of the required test signals for the CUT is derived. Both the proposed universal testing techniques do not keep track of the specific dependencies for each output and simply focus on w, the maximum number of inputs upon which any output depends. The techniques derive tests in which any output having the dependency subset of w (or less) can be pseudo-exhaustively tested. However, in general, not all outputs depend on the same number, w, of inputs. Therefore, the number of test patterns found in both techniques can be more than is necessary for pseudo-exhaustive testing.

Condensed LFSR testing based on linear codes and structural partitioning was proposed for self-testing in [14]. This technique is most effective when $w \geq n/2$. However, when $w < n/2$, more test patterns than necessary will be generated by this technique.

Another design technique using an LFSR to generate pseudo-exhaustive testing patterns was proposed in [15]. The technique is based on cyclic codes which are easier to implement and have less hardware overhead than an LFSR designed using general linear codes. However, more test patterns than necessary may be generated by this technique in some cases. For example, to design an (n, k) LFSR for an (n, w) CUT, from APPENDIX D in [16], we need to find a cyclic code that has a code length n and a minimum distance, $w + 1$. However, such a cyclic code may not exist. If a cyclic code for code length n does not exist, then we need to find a cyclic code which has next higher code length ($>n$). If the minimum distance $w + 1$ does not exist, then find a cyclic code with next higher distance $\geq w + 1$. In these worse cases, the test patterns generated by this cyclic code generator may be more than what is required for pseudo-exhaustive testing.

An efficient solution to the problem of test generation for BIST has been proposed in [1]. The polynomial-time algorithm, *two-phase cluster partitioning,* executes quickly and produces test generation circuitry of low complexity. However, in certain worst case, the algorithm may generate a sub-optimal design which requires more test patterns and/or hardware overhead. In this paper, an efficient algorithm, *advanced two-phase cluster partitioning,* is presented which is a further improvement on the original two-phase algorithm [1] to automatically design a test generator for BIST. The advanced two-phase algorithm generates less number of test signals than previous techniques and is suitable for both MTC and non-MTC circuits. The test generation procedure operates in two phases. The first phase consists of a cluster partitioning technique that itself is sufficient for circuits having the MTC property. The second phase further reduces the required number of test vectors for non-MTC circuits by forming linear sum of the test signals that were obtained at the end of the first phase. We will present detailed examples which will show the results superior to the previous techniques and also show how the techniques can improve the original two-phase algorithm.

2. PHASE ONE: CLUSTER PARTITIONING

It is well known that the same test signal can be applied to two inputs of a circuit under test (CUT) if none of the outputs is functionally dependent on both of the inputs. This fact can be used to reduce the required number of test signals. The problem of systematically reducing the number of test signals in this fashion while still exhaustively testing the CUT has been formulated as a covering problem of the cliques of a graph [10]. However, the problem of finding the cliques of a graph, a *maximal* complete sub-graphs of the original graph, is NP-complete. Thus, an efficient heuristic algorithm is needed to obtain a practical implementation for large problems.

An efficient "clique partitioning" algorithm has been proposed to solve an analogous problem of resource sharing which occurs in the area of high-level data-path synthesis [17]. (We prefer the term "cluster partitioning" since the partitions that are obtained are not necessarily cliques.) Further investigation of the heuristic algorithm in the context of data-path synthesis has shown that a smaller number of disjoint sets can be obtained if certain priorities in the algorithm are reversed. These algorithms have polynomial time complexity and as we will show, yield excellent results when applied to the test generation problem.

Before we discuss the phase one algorithm in detail, some terms used in [10] need to be reviewed:

Definition 1: An input pair (x_i, x_j) is said to be *adjacent* if there exists at least one output function f_i, $i = 1, 2, \ldots, m$ which depends on both x_i and x_j.

Definition 2: A pair (x_i, x_j) is said to be *non-adjacent* if they are not adjacent.

Definition 3: The *NA graph* is defined to have n vertices, designated by $x_1, x_2, \ldots, x_n$ corresponding to the n input lines. There is an edge between node x_i and node x_j if and only if the pair (x_i, x_j) is non-adjacent, i.e., the two inputs x_i and x_j can be tested by the same test signal.

Definition 4: The replacement of two circuit inputs by a single common test signal is represented in the graph by a *composite node*. In other words, we replace the two nodes x_i and x_j and the edge joining them with a single *composite node* labeled as (x_i, x_j).

As in Ref [10], a non-adjacency (NA) graph is used to determine which sets of circuit inputs can be connected to common test signals. Specifically, a set of input lines can all be connected to the same test signal if they form a *clique* in the NA graph. Therefore, the objective is to find the minimum number of disjoint cliques in the NA graph. However, rather than using a covering procedure, we exploit the "neighborhood property" of Ref [17] to obtain a polynomial-time procedure.

Definition 5: A node x_k in the NA graph is said to be a *common neighbor* of an edge (x_i, x_j) if edges exist from x_k to both x_i and x_j.

Definition 6: A *deleted edge* of a composite node (x_i, x_j) can arise in either of the following three ways:

(1) A node x_k which is not a common neighbor of x_i and x_j will no longer be connected to the composite node (x_i, x_j). Thus, the edge (x_i, x_k) or (x_j, x_k) has to be deleted, as appropriate.

(2) A node x_k which is a common neighbor of x_i and x_j will still be connected to the composite node (x_i, x_j). However, only one of the original two edges needs to be retained. Thus, one of the edges (x_i, x_k) or (x_j, x_k) has to be deleted. In this algorithm, the edge (x_j, x_k) will be deleted if $j > i$. Otherwise, the edge (x_i, x_k) will be deleted.

(3) The edge (x_i, x_j) itself must, of course, be deleted.

Definition 7: A *candidate pair* (x_p, x_q) in the NA graph is determined by the following criteria (listed in order of priority):

(1) (x_p, x_q) has the minimum number of deleted edges.

(2) (x_p, x_q) has the maximum number of common neighbors.

The above order of priority for pairing nodes is the *reverse* of that recommended in the "clique partitioning" approach of Ref [17]. It is found that while the above order leads to better designs in data-path synthesis problems [18], either priority scheme yields excellent results for the test generation problem.

The cluster partitioning algorithm can now be stated in the following way:

Step 1: *Establish the NA graph using the functional dependency sets F_i of the m outputs of the CUT.*

Step 2: *Traverse the list of edges in the NA graph. For each edge (x_i, x_j), compute the number of common neighbors and the number of deleted edges.*

Step 3: *Find the candidate pair (x_p, x_q). Choose the smaller of p and q as the head of the cluster. Remove the deleted edges of the composite node (x_p, x_q). Update the list of edges accordingly and recompute the numbers of common neighbors and deleted edges. If the list of edges is empty, then cluster partitioning is complete.*

Step 4: *Assume x_p is the head of the current cluster. Find a candidate pair which joins node x_p and another node, x_r. Choose the smaller of p and r as the head of the resulting cluster. Remove the deleted edges of the composite node (x_p, x_r). Update the list of edges accordingly and recompute the numbers of common neighbors and deleted edges. If the list of edges is empty, then the cluster partitioning is complete. Otherwise, if node x_p (or x_r if $r < p$) no longer appears in the updated list of edges, go to step 3 and start*

to form another cluster. Otherwise, repeat step 4 and continue to find other nodes to add to the current cluster.

The p disjoint clusters obtained through the above cluster partitioning algorithm are equivalent to the number of required test signals, i.e. the test set $T = \{s_1, s_2, \ldots, s_p\}$. Each element of T is composed of the CUT inputs which have been formed into a cluster. All inputs in the cluster s_i are represented by the same symbol x_i, which is the input in the cluster having the smallest numerical value, and the functional dependency sets are now updated accordingly. If w, the maximum number of inputs upon which any output depends is equal to p (i.e., the MTC situation) then there is no need to proceed with phase two. Otherwise, the phase two algorithm described in Section 3 must be subsequently applied to obtain the optimized test generation hardware.

In order to demonstrate the power and computational efficiency of the above cluster partitioning algorithm, we have tested it on several very large problems. In Ref. [19], several large circuits have been proposed as benchmarks for test generation algorithms. Of these, the circuits C880, C2670, C5315 and C7552 have the property that no primary output depends on all the primary inputs, so that pseudo-exhaustive testing techniques may be used. After applying the phase one algorithm to these four circuits, we find that all of them are MTC circuits. The number of test signals that are required for pseudo-exhaustively testing each of these circuits is shown in Table I. The computation times (on a Sun 3/160 workstation) required to obtain these results are also listed in the table. These circuits contain up to several thousand gates and several hundred input/output lines, yet the computation time in all cases is quite reasonable. Comparable results using the opposite priority ordering (i.e., using "clique partitioning") are also shown in Table I. As can be seen, application of either of these "data-path synthesis" procedures yields excellent results for these large examples.

3. PHASE TWO: FORMATION OF LINEAR SUM

For non-MTC circuits, the number of required test signals can be further reduced by using linear combinations of a smaller number of signals. Akers has proposed a linear sum approach for the test generation problem [12]. However, our approach differs from his in two respects. First, we do not begin to form linear sum until reductions in the number of test signals by the phase one procedure have been obtained. Second, this procedure seeks to find the *minimal* number of exclusive-or (XOR) gates which must be added. These two factors usually result in significant savings in both hardware and number of test patterns that are required to test non-MTC circuits.

In the phase two procedure [1], while looking for the test signal for the input pin e_l, we only consider the linear sum of the signals of a size two cluster C_l in the NA_l graph. If $C_l = \emptyset$ (i.e., no edges in the NA_l graph), then we add an extra test signal to the basic test set F_d as the test signal for the input pin e_l. But, in a certain case, instead of adding an extra test signal, we may use the linear sum of three or more signals in F_d as the test signal for the input pin e_l. This will indeed further reduce the number of required test signals for pseudo-exhaustive testing.

Theorem 1: A possible test signal for the input pin e_l can be the linear sum of test signals x_p and x_q, if x_p and x_q form an edge in the NA_l graph.

Proof: In step 2 of phase two algorithm, all the sets P_j's containing the element e_l are selected from the functional dependency sets. The terms $P_j - \{e_l, e_{l+1}, \ldots, e_k\}$ define sets of adjacent nodes which can be formed into an adjacency graph. Note that the non-adjacency graph is the dual graph of the adjacency graph. Thus, there does not exist any set P_j which contains all the elements x_p, x_q and e_l, if x_p and x_q form an edge in the NA_l graph. In this case, the test signal for input pin e_l can be the linear sum of test signals x_p and x_q. □

TABLE I
Results for 4 ATPG Circuits by Phase-One

Circuit Under Test					Results: Phase One		Results: Clique Partitioning	
Circuit Name	Total Gates	Total Lines	Input Lines	Output Lines	No. Test Signals	Execution Time (sec)	No. Test Signals	Execution Time (sec)
C880	383	880	60	26	45	3.3	45	3.5
C2670	1193	2670	233	140	122	120.6	122	145.2
C5315	2307	5315	178	123	67	231.6	68	222.5
C7552	3512	7552	207	108	194	14.5	194	16.2

The new phase two algorithm is listed as follows. (In the following discussion, we use p to represent the number of disjoint clusters partitioned by the phase one algorithm, and it is different from the index p representing the node x_p in the phase one algorithm.)

Step 1: *From the functional dependency sets F_i, arbitrarily choose one set F_d which has exactly w elements ($w < p$) i.e. $F_d = \{t_1, t_2, \ldots, t_w\}$, where $t_i \in T$. Then form the exclusive set $E = T - F_d$ i.e. $E = \{e_1, e_2, \ldots, e_k\}$ where $k = p - w$. Set an index $l = 1$.*

Step 2: *Traverse the functional dependency sets and find those which contain the element e_l. Let P_j denote such a set. Each set $P_j - \{e_l, e_{l+1}, \ldots, e_k\}$ contains adjacent nodes, so that these sets can be used to construct an adjacency graph. Establish the corresponding non-adjacency graph NA_l from this graph.*

Step 3: *(i) (one step look-ahead) : If $l < p - w$, choose a cluster C_l of size two in the NA_l graph which has the largest number of elements that are also members of the previous clusters C_k, $k < l$ from the graphs NA_k. (Note that these clusters are not necessarily disjoint since they are obtained from different NA graphs.) If there is more than one such cluster, then choose the cluster which will result in the maximum number of edges in the NA_{l+1} graph.*

(ii) If $l = p - w$, then arbitrarily choose any cluster C_l of size two in the NA_l graph.

(iii) If there are no edges in the NA_l graph, then we check whether there exists a test signal for the input pin e_l by formation of a linear sum of three or more signals in F_d. If it exists, then we replace the test signal for e_l by XOR of the found signals in F_d and update the functional dependency sets accordingly and go to step 5. Otherwise, we set $F_d = F_d \cup \{e_l\}$ and go to step 5.

Step 4: *Replace the test signal for e_l by the XOR of the two signals in the cluster found in the previous step. Update the functional dependency sets accordingly.*

Step 5: *If $l = p - w$, then the phase two algorithm is complete and the size of F_d is the number of required test signals. Otherwise, set $l = l + 1$ and go to step 2.*

The results of applying the combined two-phase algorithm to four non-MTC circuit examples are shown in Figures 1–3 and 4(a)—(d). Note that the number of required XOR gates is quite small in all cases. Table II compares the number of required test patterns for these four examples using this approach ("ADVANCED TWO PHASE") with the number obtained using six previously proposed test generation methods ("TWO PHASE" [1], "SDC" [10], "LFSRs/XORs" [12], "CWC" [11], "Condensed LFSR" [14] and "LFSR with Cyclic Code" [15]). As can be seen from the Table II, the advanced two-phase algorithm requires fewer test patterns in all cases.

The following is an application of the advanced two-phase algorithm to the non-MTC circuit of example 4 to demonstrate the steps of the algorithm and show how the new techniques can further reduce the number of test signals than does the original two-phase algorithm.

Example: Consider the circuit given in Figure 4(a). In step 1 of the phase one algorithm, the NA graph is established as shown in Figure 5. When the phase one algorithm is complete, the final clusters are $\{(x_1)$,

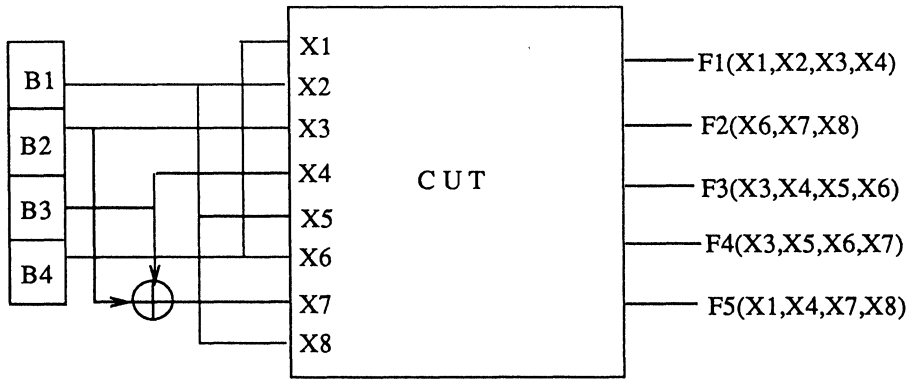

FIGURE 1 NON-MTC circuit: example 1

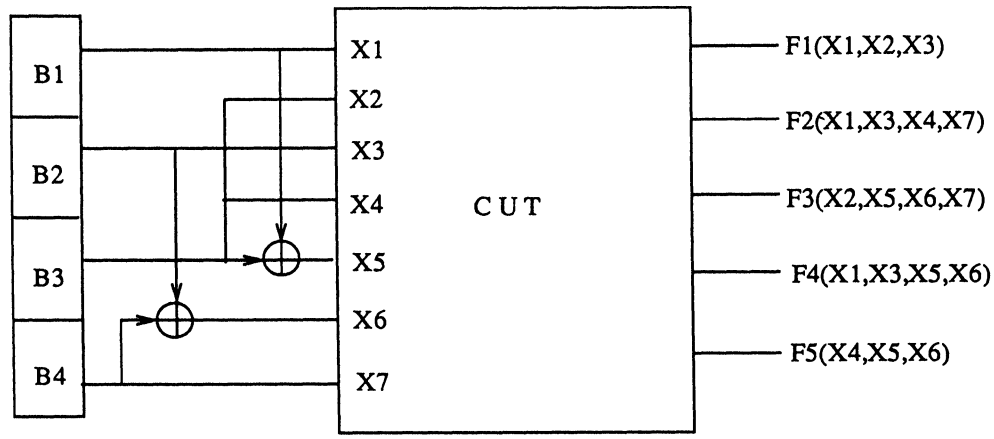

FIGURE 2 Non-MTC circuit: example 2

(x_2), (x_3), (x_4, x_5), (x_6), (x_7), $(x_8, x_9, x_{10})\}$. Therefore, $p = 7$. The test set is $T = \{x_1, x_2, x_3, x_4, x_6, x_7, x_8\}$. All the functional dependency sets are updated and listed as follows.

$$F_1 = \{x_1, x_3, x_4\}$$

$$F_2 = \{x_6, x_7, x_8\}$$

$$F_3 = \{x_2, x_7, x_8\}$$

$$F_4 = \{x_2, x_6, x_8\}$$

$$F_5 = \{x_1, x_3, x_8\}$$

$$F_6 = \{x_1, x_4, x_8\}$$

$$F_7 = \{x_3, x_4, x_8\}$$

$$F_8 = \{x_1, x_2, x_8\}$$

$$F_9 = \{x_1, x_4, x_7\}$$

$$F_{10} = \{x_2, x_3, x_7\}$$

$$F_{11} = \{x_1, x_6, x_8\}$$

$$F_{12} = \{x_4, x_6, x_8\}$$

$$F_{13} = \{x_3, x_4, x_8\}$$

$$F_{14} = \{x_4, x_7, x_8\}$$

$$F_{15} = \{x_1, x_2, x_4\}$$

But, $w = 3$ and $p \neq w$; the circuit is therefore a non-MTC circuit. So, the phase two algorithm is used. In step 1, we choose $F_d = F_1 = \{x_1, x_3, x_4\}$. The *exclusive set* $E \equiv T - F_d$ and $E = \{x_2, x_6, x_7, x_8\}$. So, $k = 4$, $e_1 = x_2$, $e_2 = x_6$, $e_3 = x_7$, and $e_4 = x_8$. Then, set $l = 1$. In step 2, traverse the functional dependency sets F_i's ($i = 1$ to 15) and find all the sets $P_1 = F_3$, $P_2 = F_4$, $P_3 = F_8$, $P_4 = F_{10}$ and $P_5 = F_{15}$, where P_j's contain the element $e_1 = x_2$. The NA_1

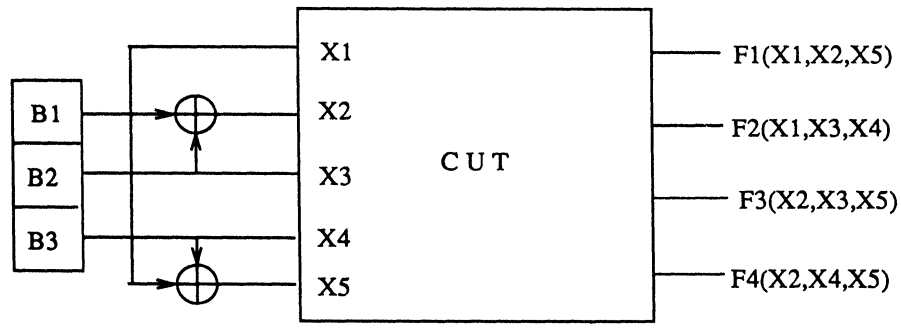

FIGURE 3 Non-MTC circuit: example 3

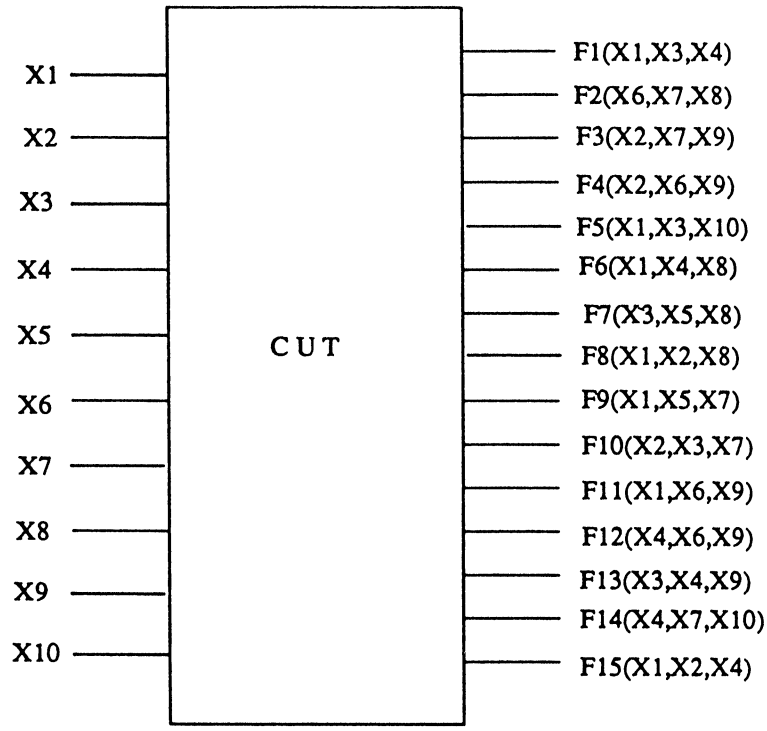

FIGURE 4(a) Non-MTC circuit: example 4

F1(X1,X3,X4)
F2(X1 ⊕ X4,X3 ⊕ X4,X1 ⊕ X3 ⊕ X4)
F3(X1 ⊕ X3,X3 ⊕ X4,X1 ⊕ X3 ⊕ X4)
F4(X1 ⊕ X3,X1 ⊕ X4,X1 ⊕ X3 ⊕ X4)
F5(X1,X3,X1 ⊕ X3 ⊕ X4)
F6(X1,X4,X1 ⊕ X3 ⊕ X4)
F7(X3,X4,X1 ⊕ X3 ⊕ X4)
F8(X1,X1 ⊕ X3,X1 ⊕ X3 ⊕ X4)
F9(X1,X4,X3 ⊕ X4)
F10(X1 ⊕ X3,X3,X3 ⊕ X4)
F11(X1,X1 ⊕ X4,X1 ⊕ X3 ⊕ X4)
F12(X4,X1 ⊕ X4,X1 ⊕ X3 ⊕ X4)
F13(X3,X4,X1 ⊕ X3 ⊕ X4)
F14(X4,X3 ⊕ X4,X1 ⊕ X3 ⊕ X4)
F15(X1,X1 ⊕ X3,X4)

FIGURE 4(b) Formation of linear sum in output dependency sets

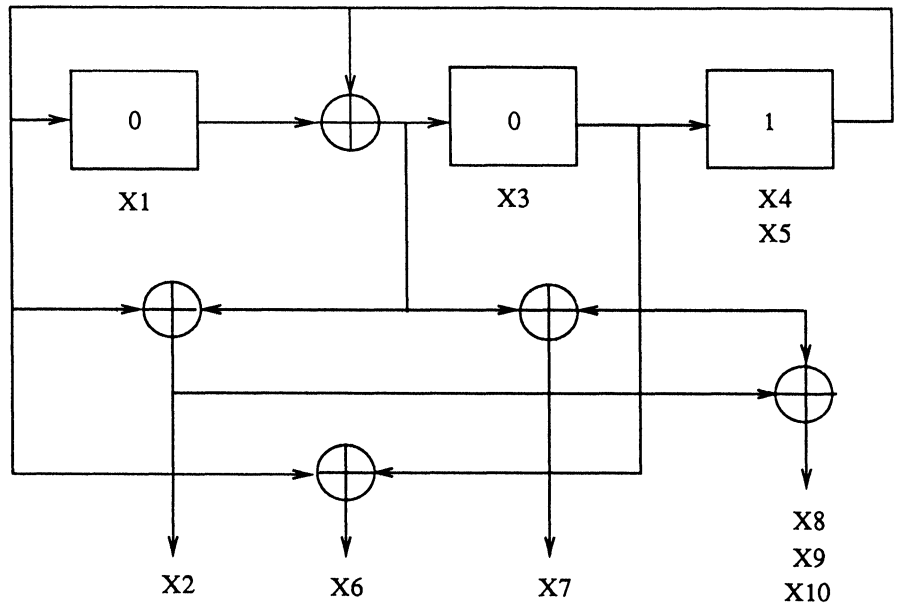

FIGURE 4(c) Test generator by Advanced Two-Phase

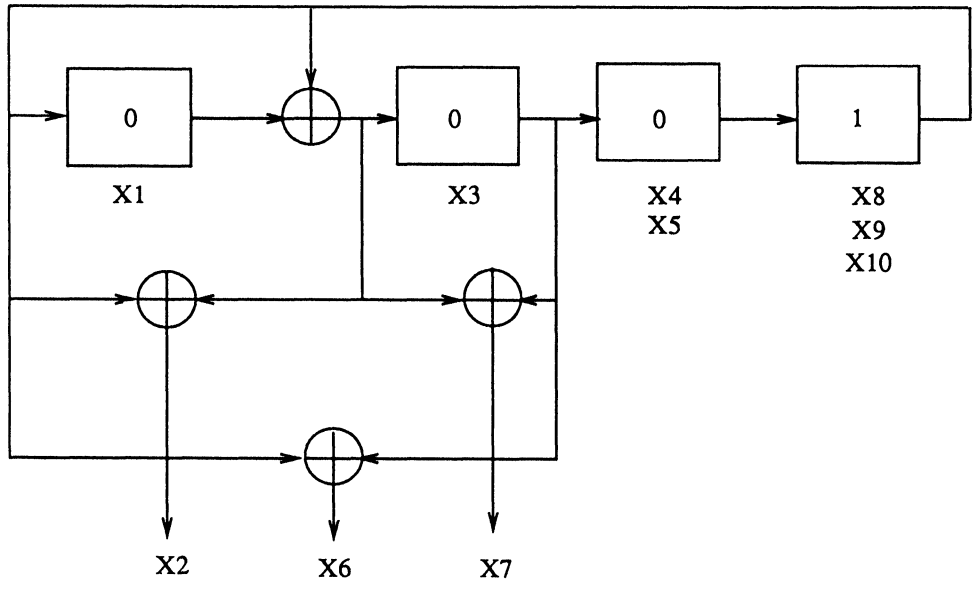

FIGURE 4(d) Test generator by Two-Phase [1]

TABLE II
Number of Test Patterns in Comparison with Previous Techniques

Circuit	ADVANCED TWO PHASE	TWO PHASE	SDC	LFSRs /XORs	CWC	Condensed LFSR	LFSR with Cyclic code
Example 1	16	16	31	63	16	31	63
Example 2	16	16	63	63	21	31	63
Example 3	8	8	31	15	10	15	15
Example 4	8	15	127	15	14	63	15

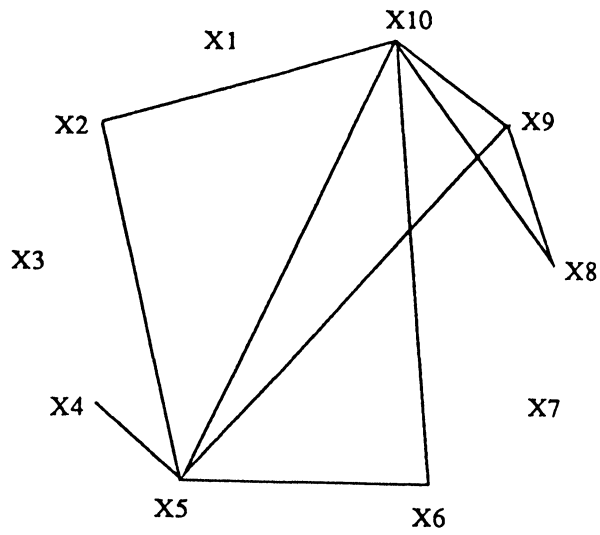

FIGURE 5 The *NA* graph of Example 4

graph is established by the following sets and shown in Figure 6(a).

$$P_1 - \{x_2, x_6, x_7, x_8\} = \emptyset$$

$$P_2 - \{x_2, x_6, x_7, x_8\} = \emptyset$$

$$P_3 - \{x_2, x_6, x_7, x_8\} = \{x_1\}$$

$$P_4 - \{x_2, x_6, x_7, x_8\} = \{x_3\}$$

$$P_5 - \{x_2, x_6, x_7, x_8\} = \{x_1, x_4\}$$

From the NA_1 graph, we choose $C_1 = \{x_1, x_3\}$. Since $C_1 \neq \emptyset$, we execute step 4. The test signal for input pin x_2 can be the XOR of the test signals for input pins x_1 and x_3 (i.e. $x_2 = x_1$ XOR x_3). Substitute x_1 and x_3 for x_2 in all functional dependency sets F_i's. Therefore,

$$F_1 = \{x_1, x_3, x_4\}$$

$$F_2 = \{x_6, x_7, x_8\}$$

$$F_3 = \{x_1, x_3, x_7, x_8\}$$

$$F_4 = \{x_1, x_3, x_6, x_8\}$$

$$F_5 = \{x_1, x_3, x_8\}$$

$$F_6 = \{x_1, x_4, x_8\}$$

$$F_7 = \{x_3, x_4, x_8\}$$

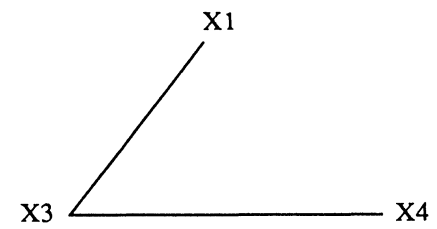

FIGURE 6(a) The NA_1 graph of example 4

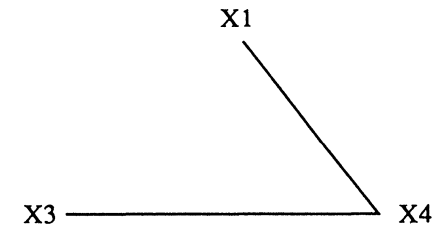

FIGURE 6(b) The NA_2 graph of example 4

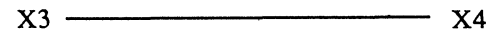

FIGURE 6(c) The NA_3 graph of example 4

$$F_8 = \{x_1, x_3, x_8\}$$

$$F_9 = \{x_1, x_4, x_7\}$$

$$F_{10} = \{x_1, x_3, x_7\}$$

$$F_{11} = \{x_1, x_6, x_8\}$$

$$F_{12} = \{x_4, x_6, x_8\}$$

$$F_{13} = \{x_3, x_4, x_8\}$$

$$F_{14} = \{x_4, x_7, x_8\}$$

$$F_{15} = \{x_1, x_3, x_4\}$$

In step 5, $l = 1 \neq p - w$, phase two algorithm is not complete. We set $l = 2$ and go to step 2.

In step 2, traverse the functional dependency sets F_i's ($i = 1$ to 15) and find all the sets $P_1 = F_2$, $P_2 = F_4$, $P_3 = F_{11}$, and $P_4 = F_{12}$, where P_j's contain the element $e_2 = x_6$. The NA_2 graph is established by the following sets. This is shown in Figure 6(b).

$$P_1 - \{x_6, x_7, x_8\} = \emptyset$$

$$P_2 - \{x_6, x_7, x_8\} = \{x_1, x_3\}$$

$$P_3 - \{x_6, x_7, x_8\} = \{x_1\}$$

$$P_4 - \{x_6, x_7, x_8\} = \{x_4\}$$

From the NA_2 graph, we have two possible resulting clusters $\{x_1, x_4\}$ or $\{x_3, x_4\}$. Both of the clusters have the same number (1) of elements in the previous cluster $C_1 = \{x_1, x_3\}$. So, more than one cluster can be the candidate. We need to choose the cluster which will result in the maximum number of edges in the NA_3 graph. If we choose $C_2 = \{x_1, x_4\}$, then in step 4 the test signal for the input pin x_6 can be the XOR of the test signals for input pins x_1 and x_4 (i.e. $x_6 = x_1$ XOR x_4). Substitute x_1 and x_4 for x_6 in all functional dependency sets F_i's. Therefore,

$$F_1 = \{x_1, x_3, x_4\}$$

$$F_2 = \{x_1, x_4, x_7, x_8\}$$

$$F_3 = \{x_1, x_3, x_7, x_8\}$$

$$F_4 = \{x_1, x_3, x_4, x_8\}$$

$$F_5 = \{x_1, x_3, x_8\}$$

$$F_6 = \{x_1, x_4, x_8\}$$

$$F_7 = \{x_3, x_4, x_8\}$$

$$F_8 = \{x_1, x_3, x_8\}$$

$$F_9 = \{x_1, x_4, x_7\}$$

$$F_{10} = \{x_1, x_3, x_7\}$$

$$F_{11} = \{x_1, x_4, x_8\}$$

$$F_{12} = \{x_1, x_4, x_8\}$$

$$F_{13} = \{x_3, x_4, x_8\}$$

$$F_{14} = \{x_4, x_7, x_8\}$$

$$F_{15} = \{x_1, x_3, x_4\}$$

we can establish the NA_3 graph shown in Fig. 6(c) (the details for establishing NA_3 graph will be given below), and we find there is one edge in the graph.

However, if we choose $C_2 = \{x_3, x_4\}$, then in step 4 the test signal for the input pin x_6 can be the XOR of the test signals for input pins x_3 and x_4 (i.e. $x_6 = x_3$ XOR x_4). Substitute x_3 and x_4 for x_6 in all functional dependency sets F_i's. Therefore,

$$F_1 = \{x_1, x_3, x_4\}$$

$$F_2 = \{x_3, x_4, x_7, x_8\}$$

$$F_3 = \{x_1, x_3, x_7, x_8\}$$

$$F_4 = \{x_1, x_3, x_4, x_8\}$$

$$F_5 = \{x_1, x_3, x_8\}$$

$$F_6 = \{x_1, x_4, x_8\}$$

$$F_7 = \{x_3, x_4, x_8\}$$

$$F_8 = \{x_1, x_3, x_8\}$$

$$F_9 = \{x_1, x_4, x_7\}$$

$$F_{10} = \{x_1, x_3, x_7\}$$

$$F_{11} = \{x_1, x_3, x_4, x_8\}$$

$$F_{12} = \{x_3, x_4, x_8\}$$

$$F_{13} = \{x_3, x_4, x_8\}$$

$$F_{14} = \{x_4, x_7, x_8\}$$

$$F_{15} = \{x_1, x_3, x_4\}$$

where F_2, F_3, F_9, F_{10} and F_{14} have the element $e_3 = x_7$. Following the phase two algorithm, we establish the NA_3 graph and find that there are no edges in the graph.

Therefore, in step 3 of phase two algorithm, we choose $C_2 = \{x_1, x_4\}$, which is better than $\{x_3, x_4\}$ in BIST design (see the comparison below).

After we choose $C_2 = \{x_1, x_4\}$, in step 5, $l = 2 \neq p - w$, phase two algorithm is not complete. We set $l = 3$ and go to step 2.

In step 2, traverse the functional dependency sets F_i's ($i = 1$ to 15) and find all the sets $P_1 = F_2$, $P_2 = F_3$, $P_3 = F_9$, $P_4 = F_{10}$ and $P_5 = F_{14}$, where the P_j's contain the element $e_3 = x_7$. The NA_3 graph is established by the following sets and shown in Figure 6(c).

$$P_1 - \{x_7, x_8\} = \{x_1, x_4\}$$

$$P_2 - \{x_7, x_8\} = \{x_1, x_3\}$$

$$P_3 - \{x_7, x_8\} = \{x_1, x_4\}$$

$$P_4 - \{x_7, x_8\} = \{x_1, x_3\}$$

$$P_5 - \{x_7, x_8\} = \{x_4\}$$

From the NA_3 graph, $C_3 = \{x_3, x_4\}$ is the only cluster found with a size of 2. Since $C_3 \neq \emptyset$, we execute step 4. The test signal for input pin x_7 can be the XOR of the test signals for input pins x_3 and x_4 (i.e. $x_7 = x_3$ XOR x_4). Substitute x_3 and x_4 for x_7 in all functional dependency sets F_i's. Therefore,

$$F_1 = \{x_1, x_3, x_4\}$$

$$F_2 = \{x_1, x_3, x_4, x_8\}$$

$$F_3 = \{x_1, x_3, x_4, x_8\}$$

$$F_4 = \{x_1, x_3, x_4, x_8\}$$

$$F_5 = \{x_1, x_3, x_8\}$$

$$F_6 = \{x_1, x_4, x_8\}$$

$$F_7 = \{x_3, x_4, x_8\}$$

$$F_8 = \{x_1, x_3, x_8\}$$

$$F_9 = \{x_1, x_3, x_4\}$$

$$F_{10} = \{x_1, x_3, x_4\}$$

$$F_{11} = \{x_1, x_4, x_8\}$$

$$F_{12} = \{x_1, x_4, x_8\}$$

$$F_{13} = \{x_3, x_4, x_8\}$$

$$F_{14} = \{x_3, x_4, x_8\}$$

$$F_{15} = \{x_1, x_3, x_4\}$$

In step 5, $l = 3 \neq p - w$, the phase two algorithm is not complete. We set $l = 4$ and go to step 2.

In step 2, traverse the functional dependency sets F_i's ($i = 1$ to 15) and find all the sets $P_1 = F_2$, $P_2 = F_3$, $P_3 = F_4$, $P_4 = F_5$, $P_5 = F_6$, $P_6 = F_7$, $P_7 = F_8$, $P_8 = F_{11}$, $P_9 = F_{12}$, $P_{10} = F_{13}$ and $P_{11} = F_{14}$, where the P_j's contain the element $e_4 = x_8$. The NA_4 graph can be established by the sets $P_j - \{x_8\}$. We find that there are no edges in the NA_4 graph. $C_4 = \emptyset$. Using the original phase two procedure [1], because $C_4 = \emptyset$, we consider that the test signal for input pin x_8 can not be a linear sum of the test signals in the current test $F_d = \{x_1, x_3, x_4\}$. Therefore, we need to add an extra test signal x_8 to the current test set F_d, i.e., $F_d = F_d \cup \{x_8\} = \{x_1, x_3, x_4, x_8\}$ and go to step 5. But, this design procedure leaves a better solution out of consideration and generates a non-optimal solution. In the new phase two procedure, while traversing the test signals in F_d, we find that the test signal for input pin x_8 can be a linear sum of the test signals for input pins x_1, x_3 and x_4 (i.e. $x_8 = x_1$ XOR x_3 XOR x_4) (we may confirm this in Figure 4(b)). No extra test signal is needed to add to the basic test set F_d. Then, we substitute x_1, x_3 and x_4 for x_8 in all functional dependency sets and go to step 5.

In step 5, $l = 4 = p - w$, so the phase two cluster partitioning algorithm is complete. The size of F_d is three. Therefore, three test signals are capable of pseudo-exhaustively testing the circuit. The test generator is shown in Fig. 4(c). But, using the original two-phase algorithm, we need four test signals in the test generator as shown in Fig. 4(d).

Using the original two-phase algorithm, if we were to choose $C_2 = \{x_3, x_4\}$ as the candidate in the NA_2 graph, we would find that there are no edges in NA_3 graph. Therefore, an extra test signal for the input pin x_7 is needed, i.e., $F_d = F_d \cup \{x_7\} = \{x_1, x_3, x_4, x_7\}$. Continuing to follow the phase two algorithm, we'll find that there will be no edges in the NA_4 graph. Therefore, an extra test signal for the input pin x_8 is needed and is added to the test set F_d, i.e., $F_d = F_d \cup \{x_8\} = \{x_1, x_3, x_4, x_7, x_8\}$. The size of F_d is five. Therefore, five test signals are needed, a number greater than that of test signals required by the proposed design in Figure 4(c).

4. SIMULATION RESULTS

Based on *Advanced Two-Phase Cluster Partitioning* Algorithm, a design generator named *BISTSYN* has been developed and implemented to facilitate the BIST design. The input to the design generator can be either a circuit description at the gate level which is viewed as a netlist or the circuit output functional dependency sets. *BISTSYN* provides the BIST mechanisms as the output.

In this Section, we use 42 circuits to evaluate "BISTSYN" and also compare the generated results with those produced by five previously proposed test generation methods ("SDC" [10], "LFSRs/XORs" [12], "CWC" [11], "Condensed LFSR" [14] and "LFSR with Cyclic Code" [15]). Table III compares the number of required test patterns for all these 42

TABLE III
Number of Test Patterns in Comparison with Previous Techniques

Number of Inputs	Number of Outputs	W	P	SDC	LFSR/XOR	CWC	Condensed LFSR	LFSR with Cyclic Code	BISTSYN
5	5	3	3	8	15	8	8	8	8
5	5	4	4	16	16	16	16	63	16
5	5	5	5	32	32	32	32	63	32
8	8	2	2	4	15	4	4	7	4
8	8	3	4	15	15	8	15	8	8
8	8	4	4	16	31	16	16	63	16
8	8	5	5	32	63	32	32	63	32
8	8	6	6	64	127	64	64	64	64
8	8	7	8	255	128	128	255	2047	128
10	10	2	2	4	15	4	4	7	4
10	10	3	3	8	15	8	8	8	8
10	10	4	5	31	63	16	16	63	16
10	10	5	5	32	63	32	32	63	32
10	10	6	7	127	127	64	64	64	64
10	10	7	9	511	255	170	255	2047	128
10	10	9	10	1023	512	512	1023	2047	512
10	10	10	10	1024	1024	1024	1024	2047	1024
10	15	2	3	7	15	4	4	7	4
10	15	3	4	15	31	8	8	8	8
10	15	4	5	31	63	16	16	63	16
10	15	5	7	127	63	42	63	63	32
10	15	6	9	511	127	120	255	1023	64
10	15	7	9	511	255	170	255	2047	128
10	15	8	9	511	511	256	256	2047	256
10	15	9	10	1023	512	512	512	2047	512
15	10	2	2	4	15	4	4	7	4
15	10	3	3	8	31	8	8	8	8
15	10	4	4	16	63	16	16	63	16
15	10	5	5	32	127	32	32	63	32
15	10	6	6	64	255	64	64	64	64
15	10	7	7	2047	511	330	1023	1047	128
15	10	9	11	2047	2047	682	1023	2047	512
15	10	10	12	4095	2047	1365	2047	2047	1024
15	15	2	3	7	15	4	4	7	4
15	15	3	4	15	31	8	8	8	8
15	15	4	4	16	63	16	16	63	16
15	15	5	6	63	127	32	32	63	32
*15	15	6	8	255	255	36	127	1023	64
15	15	7	13	8191	511	572	2047	1023	255
15	15	8	9	511	1023	256	256	2047	256
15	15	9	12	4095	2047	992	2047	2047	512
15	15	10	12	4095	2047	1365	2047	2047	1024

circuits. As seen from the Table III, "BISTSYN" requires fewer test patterns in almost all circuits except the one marked with "*" in which "CWC" requires 36 test patterns while "BISTSYN" needs 64 test patterns.

For some benchmarks, the size of pseudo-exhaustive test generator may be still relatively large even the reduction of the number of required test signals by *Advanced Two-Phase Cluster Partitioning* algorithm is applied. For these type of circuits, the fault

TABLE IV
Fault Simulation Results of Benchmark

		Embedding BIST						*BISTSYN*					
Circuit	PI	Faults	Counter size	Generated Tests	Undetected Faults	Fault Coverage	Fault Efficiency	Faults	LFSR size	Generated Tests	Undetected Faults	Fault Coverage	Fault Efficiency
c432	36	524	10	1024	10	98.09%	98.85%	524	36	4999	4	99.24%	99.81%
c499	41	758	10	1024	11	98.55%	99.60%	758	41	798	8	98.95%	100.00%
c880	60	942	13	8192	0	100.00%	100.00%	942	45	15456	0	100.00%	100.00%
c1355	41	1574	12	4096	8	99.49%	100.00%	1574	41	10000	8	99.49%	100.00%
c1908	33	1879	13	8192	11	99.41%	99.90%	1879	33	4999	9	99.52%	100.00%
c2670	233	2747	16	65536	398	85.51%	89.32%	2747	122	824113	149	94.57%	98.78%
c3540	50	3428	13	8192	266	92.24%	96.08%	3428	50	99999	137	96.00%	100.00%
c5315	178	5350	13	8192	80	98.50%	99.60%	5350	67	4999	59	98.88%	100.00%
c6288	32	7744	9	512	83	98.93%	99.36%	7744	32	256	34	99.56%	100.00%
c7552	207	7550	N/A	N/A	N/A	N/A	N/A	7550	194	499999	202	97.33%	98.25%
Average	91.1	3250	12.11	11662	96.3	96.75%	98.07%	3250	66	146562	61	98.35%	99.68%

simulation based on the designed pseudo-exhaustive test generator is performed. Table IV compares the fault simulation results for these benchmarks. The fault coverage is calculated using the number of "equivalent" single stuck-at faults and the fault efficiency is calculated as the percentage of detected faults out of all detectable faults. In Table IV, the first column are results from "Embedding BIST" [20] and the second column are the fault simulation results from *BISTSYN*. As shown in Table IV, "BISTSYN" achieves very high fault coverage and fault efficiency in all the circuits after a reasonable number of test patterns are applied from the designed pseudo-exhaustive test generator.

5. SUMMARY

An efficient technique for designing a test generator to pseudo-exhaustively test a circuit in which none of the outputs depend on all of the inputs has been presented. The algorithm consists of phase one and phase two procedures. Using only phase one, a minimum number of test signals required for pseudo-exhaustively testing a CUT without linear sum is found. For a CUT with the non-MTC property, the advanced phase two algorithm can be subsequently applied to further reduce the number of test signals that are obtained by original phase-two procedure [1] in face of the worst case. Although this technique requires fewer test vectors and/or lower hardware overhead compared to previous methods, it has the property of using the sequential approach for formation of linear sum (i.e., generating the linear sum in sequence). Therefore, the algorithm is a global hierarchical design. The sequence of design decisions that are made in this design hierarchy are based on assumptions about what can be achieved at the current point in the design process. Therefore, these decisions are locally optimal, and may still possible generate a suboptimal design at the final state. For these problems, we seek to develop an alternate approach by which accurate predictions about the implications of design decisions can be made. A global search for linear sum is currently under investigation.

Acknowledgement

This work was supported in part by the Ohio State Research Challenge Award 660808 and the Graduate School of the Wright State University. Special thanks are due to Professor G.E. Sobelman for valuable discussions of the original two-phase algorithm.

References

[1] C.-I.H. Chen and G.E. Sobelman, "An Efficient Approach to Pseudo-Exhaustive Test Generation for BIST Design," *Proc. IEEE Int. Conf. on Computer Design*, pp. 576–579, 1989.

[2] E.J. McCluskey, "Built-In Self Test Techniques," *IEEE Design and Test of Computers,* pp. 21–28, Apr. 1985.
[3] V.K. Agrawal and E. Cerney, "Store and Generate Built-In Testing Approach," *Proc. 11th Int. Symp. Fault-Tolerant Computing,* pp. 35–40, 1981.
[4] D. Komonystky, "LSI Self-Test Using LSSD and Signature Analysis," *Proc. Int. Test Conf.,* pp. 414–424, 1982.
[5] Y.M. El-Zig, "S_3: VLSI Self-Testing Using a Signature Analysis and Scan-Path Techniques," *Proc. Inst. Conf. on Computer-Aided Design,* pp. 73–76, 1983.
[6] P.H. Bardell and W.H. McAnney, "Self-Testing of Multichip Logic Modules," *Proc. Int. Test Conf.,* pp. 200–204, 1982.
[7] P.P. Fasang, "BIDCO, Built-In Digital Circuit Observer," *Proc. Int. Test Conf.,* pp. 261–266, 1980.
[8] A. Kransniewski and A. Albicki, "Automatic Design of Exhaustively Self-Testing Chips with BIBLO Modules," *Proc. Int. Test Conf.,* pp. 362–371, 1985.
[9] B. Koenemann et al., "Built-In Logic Block Observation Techniques," *Proc. Int. Test Conf.,* pp. 37–41, 1979.
[10] Z. Barzilai, J. Savir, G. Markowsky, and M.G. Smith, "The Weighted Syndrome Sums Approach to VLSI Testing," *IEEE Trans. on Comp.,* Vol. C-29, pp. 1012–1013, Nov. 1981.
[11] E.J. McCluskey, "Verification Testing—A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers,* Vol. C-33, No. 6, June 1984.
[12] S.B. Akers, "On the Use of Linear Sums in Exhaustive Testing," *Proc. 15th Fault Tolerant Comp. Symp.,* pp. 148–153, June 1985.
[13] N. Vasanthavada and P.N. Marinos, "An Operationally Efficient Scheme for Exhaustive Test-Pattern Generation Using Linear Codes," *Proc. IEEE Test Conf.,* pp. 476–482, 1985.
[14] L.T. Wang and E.J. McCluskey, "Condensed Linear Feedback Shifter Register (LFSR) Testing—A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers,* Vol. C-35, No. 4, 1986.
[15] L.T. Wang and E.J. McCluskey, "Circuits for Pseudo-exhaustive Test Pattern Generation," *Proc. IEEE Test Conf.,* pp. 25–37, 1986.
[16] W.W. Peterson and E.J. Weldon, Jr., *Error Correcting Codes,* 2nd Edition, Cambridge, MA: M.I.T. Press, 1972.
[17] C.J. Tseng and D.P. Siewiorek, "Automated Synthesis of Data Paths in Digital System," *IEEE Trans. on Computer-Aided Design,* Vol. CAD-5, No. 3, July 1986.
[18] C.-I.H. Chen and G.E. Sobelman, "Cluster Partitioning Techniques for Data Path Synthesis," to appear in *VLSI Design: An International Journal of Computer-Chip Design, Simulation, and Testing.*
[19] F. Brglez, P. Pownall and R. Hum, "Accelerated ATPG and fault grading via testability analysis," *Proc. IEEE Int. Symp. Circuits and Systems,* pp. 695–698, 1985.
[20] S.B. Akers and W. Jansz, "Test Set Embedding in a Built-In Self-Test Environment," *Proc. 1989 International Test Conference,* pp. 257–263, 1989.

Biography

CHIEN-IN HENRY CHEN is an Assistant Professor of Electrical Engineering at Wright State University. He received the B.S. degree from the National Taiwan University, Taiwan, in 1981, the M.S. degree from the University of Iowa, Iowa City, in 1986, and the Ph.D. degree from the University of Minnesota, Minneapolis, in 1989, all in the Electrical Engineering. During the summer 1992, he worked as a Faculty Research Associate at Wright Laboratory, Wright-Patterson Air Force Base, Dayton, Ohio. His current research interests are in areas of computer-aided design, simulation and testing of VLSI circuits, design for testability, and fault-tolerant computing.