Combinational Automatic Test Pattern Generation for Acyclic Sequential Circuits

Yong Chang Kim, Vishwani D. Agrawal, and Kewal K. Saluja

***Abstract*—It is known that the complexity of automatic test pattern generation (ATPG) for acyclic sequential circuits is similar to that of combinational ATPG. The general problem, however, requires time-frame expansion and multiple-fault detection and hence does not allow the use of available combinational ATPG programs. The first contribution of this work is a combinational single-fault ATPG method for the most general class of acyclic sequential circuits. Without inserting any real hardware, we create a functionally equivalent "balanced" ATPG model of the circuit in which all reconverging paths have the same sequential depth. Some primary inputs and gates are duplicated in this model, which is converted into a combinational circuit by shorting all flip-flops. A test vector obtained by a combinational ATPG program for a fault in this combinational circuit is transformed into a test sequence to detect a corresponding fault in the original sequential circuit. A combinational ATPG program finds tests for all but a small set of faults that must be explicitly detected as multiple-faults. Those are modeled for ATPG using the second contribution of this work, which is a generalized method to model any given multiple stuck-at fault as a single stuck-at fault. The procedure requires insertion of at most $n + 3$ modeling gates for a fault of multiplicity n. We show that the modeled circuit is functionally equivalent to the original circuit and the targeted multiple fault is equivalent to the modeled single stuck-at fault. Benchmark results show at least an order of magnitude saving in the ATPG CPU time by the new combinational method over sequential ATPG.**

***Index Terms*—Acyclic sequential circuit, automatic test pattern generation (ATPG), balanced model, combinational test generation, design for test (DFT), partial scan, test.**

I. Introduction

For combinational circuits, algorithms and programs exist that provide acceptable fault coverages with 100% fault efficiency. The same performance has not been possible for sequential automatic test pattern generation (ATPG). This is the main reason for the widespread acceptance of the full-scan design [8]. However, concerns about the full-scan overheads of area, delay and test time have motivated designers and researchers to explore partial-scan techniques [1].

Early attempts at purely combinational ATPG-based partial scan design produced overheads that were often close to those of full-scan [2]. A low overhead partial-scan technique, in which scan flip-flops (FFs) break feedback paths, was proposed by Cheng and Agrawal [5], and Kunzmann and Wunderlich [25]. The resulting acyclic sequential circuit is guaranteed to be initializable and has a well-defined sequential depth that bounds the complexity of sequential ATPG [4]. Using an analysis by Miczo [26], showing that the ATPG complexity of acyclic circuits is similar to combinational circuit, a combinational ATPG method for a single output circuit was proposed by Kunzmann and Wunderlich [25]. Inoue *et al.* [18], [19] give a constructive definition of a balanced register transfer level (RTL) model for multiple output acyclic sequential circuits called the time expansion model (TEM). The TEM is nonunique and hence can be nonoptimal. The literature does not give an algorithm to derive an optimal TEM. Being at RTL, it cannot guarantee a 100% fault efficiency for gate-level faults.

Manuscript received August 28, 2003; revised December 8, 2003 and July 2, 2004. This work was supported in part by the National Science Foundation under Grant MIP9714034. This paper was recommended by Associate Editor N. K. Jha.

Y. C. Kim is with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright Patterson AFB, OH 45433 USA (e-mail: yong.kim@afit.edu).

V. D. Agrawal is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: vagrawal@eng.auburn.edu).

K. K. Saluja is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 USA (e-mail: saluja@engr.wisc.edu).

Digital Object Identifier 10.1109/TCAD.2005.847894

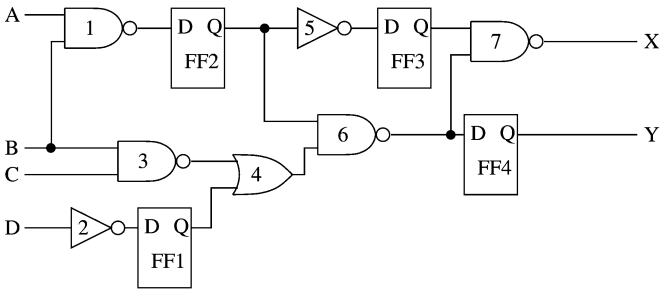

Fig. 1. Multioutput acyclic circuit.

The focus of our paper is efficient derivation of tests for general acyclic sequential circuits using any conventional single-fault combinational ATPG program. Starting with the ideas of Miczo [26] and Gupta *et al.* [11], [13], we develop a combinational model suitable for combinational test generation. We can then detect all testable faults and identify all untestable faults [22]–[24] for acyclic sequential circuits using combinational ATPG. The application domain of this paper is fully synchronous and acyclic circuits. The paper makes two important contributions: 1) a combinational test generation method for acyclic sequential circuits in Section III and 2) a generalized single fault model for multiple faults in Section IV.

II. Circuit Subclasses and ATPG Methods

A clocked synchronous circuit is called *acyclic* if the FFs form no cycle. An acyclic circuit is known to be *initializable* and the test sequence length of any detectable single fault has an upper bound $d_{\rm seq} + 1$, where $d_{\rm seq}$ is the *sequential depth*, defined as the maximum number of FFs on the longest path between primary inputs (PIs) and primary outputs (POs) in the circuit [4]. Fig. 1 shows a multioutput acyclic circuit example used in the subsequent discussion.

Miczo [26] gives an analysis of acyclic circuits and shows that the test generation problem for acyclic sequential circuit is similar to the combinational ATPG. Following up on the idea, Min and Rogers [27] obtained a simple combinational ATPG model in which all FFs are shorted during test generation. A test vector repeated $d_{\rm seq}$ times detects the fault in the sequential circuit. This procedure, however, leaves many faults undetected and one needs a sequential ATPG to achieve 100% fault efficiency.

Miczo [26], Kunzmann and Wunderlich [25], and Inoue *et al.* [18], [19] point out that an exact combinational ATPG model need not duplicate the entire combinational logic $d_{\rm seq}$ times. Although these methods define the required structure for combinational test generation, no implementable algorithms for generating such models are provided. Another remaining problem is that the detection of a single fault in the original circuit requires detecting multiple faults in the logic that is duplicated.

Gupta *et al.* [11], [13] defined a subclass of acyclic sequential circuits called "balanced" that avoided the test generation for multiple faults. In a balanced (BAL) circuit, all signal paths between any pair of nodes (PIs, POs, FFs, or gates) have the same number of FFs. Shorting FFs in a balanced circuit generates a combinational model in which all single faults of the original circuit are mapped as single faults. This model can produce tests for all detectable faults. Any acyclic circuit can be converted into a balanced circuit by test-mode isolation of a selected set of

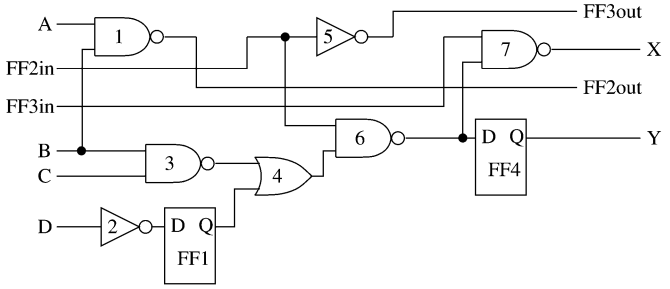

Fig. 2. Balanced (BAL) partial-scan circuit for the acyclic circuit of Fig. 1.

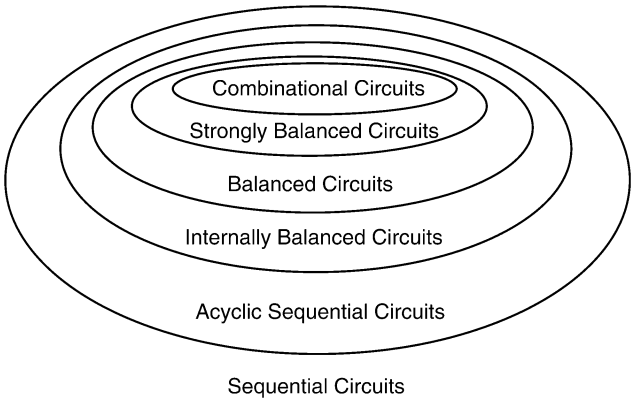

Fig. 3. Subclasses of sequential circuits.

FFs via a partial scan. For example, the circuit of Fig. 1 has fanouts at PI B and at the output of FF2 that reconverge with different numbers of FFs (unbalanced depths). When FF2 and FF3 are scanned, the circuit is balanced and its balanced model is shown in Fig. 2. Note that using functional information, it may be possible to obtain a combinational model of a sequential circuit without balancing the circuit, such as in the case of switched balanced model [12], in which the circuit is balanced except for fanouts of *known* multiplexers, or sequential circuit with feedback loops [7], [14], [24].

Fujiwara *et al.* [9], [10], [16], [17], [31] introduced another subclass of acyclic circuits called internally balanced (IB) circuits. In an IB circuit, all node-pairs, except those involving a primary input, are balanced. This requires a lower partial-scan overhead than a balanced circuit. A combinational model is generated by replacing FFs with wires or buffers and the PIs with unbalances are split as additional PIs. Tests are generated for this model using combinational ATPG. Each combinational vector is converted into a test sequence of length $d_{\rm seq}+1$. The bits from a split PI are appropriately placed in the sequence for application to the corresponding original PI. The bits of unsplit PIs are simply duplicated in each vector. An IB circuit model of the circuit of Fig. 1 is obtained by scanning FF3, splitting unbalanced fanouts PI B, then shorting the remaining FFs.

In order to compact test sequences, Balakrishnan and Chakradhar [3] proposed the strongly balanced (SB) subclass that is a more restrictive structure than the balanced circuit. A strongly balanced circuit is balanced and, in addition, requires that all paths between a node and all PIs have the same sequential depth. The additional constraint allows the combinational vectors to be pipelined. The test sequence is generated by concatenating combinational vectors and repeating only the last vector $d_{\rm seq}$ times. The strongly balanced structure uses a combinational ATPG to derive tests that are significantly more compact than those for either the internally balanced or the balanced circuit, but more FFs than the other two subclasses may be scanned to make the circuit strongly balanced. For the example circuit of Fig. 1, we scan three FFs, FF1, FF2, and FF3, to make it strongly balanced (SB).

In the full-scan method [8], all FFs in the circuit are scanned to make the circuit combinational.

Starting from a general sequential circuit, we can obtain five subclasses of acyclic circuits by progressively scanning more FFs. Associated with each subclass there is a combinational ATPG method. Fig. 3 shows the set relationships: Combinational (Cmb.) $\subseteq$ strongly balanced (SB) $\subseteq$ balanced (BAL) $\subseteq$ internally balanced (IB) $\subseteq$ acyclic sequential (Acy.) $\subseteq$ sequential circuits.

III. New Combinational ATPG Method

Using the idea that the circuit should be balanced [13] for combinational ATPG to be effective, we create a balanced ATPG model (BAM) for any given acyclic circuit. All unbalanced fanouts, i.e., fanouts reconverging with different sequential depths, are moved toward primary inputs using a retiming-like transformation. Such fanouts can supply different values to branches in different time frames. Unbalanced PI fanouts are split as additional primary inputs and all FFs are replaced with buffers to produce a combinational ATPG (C-ATPG) circuit. A combinational test vector for a fault in the C-ATPG circuit is generated using a combinational ATPG program and converted into a vector sequence that detects the corresponding fault in the original acyclic sequential circuit. Using a fault-mapping algorithm, we classify the undetected faults in this model as either untestable or multiply-testable. The latter, typically consisting of less than 5% of all faults, are modeled as special single faults in the C-ATPG circuit. Our test generation method has two steps, balanced ATPG model generation and test generation.

A. BAM Generation

In a sequential circuit, unbalanced paths often carry signals to a reconvergent gate. These signals originate at PIs in different time frames. In order to allow combinational ATPG the choice of assigning different values to a signal in different time frames an unbalanced fanout signal should be duplicated. Algorithm 1 generates a combinational ATPG (C-ATPG) circuit.

In Algorithm 1, our goal is to assign a unique weight to each node with respect to each output, where a node is PI, gate, FF, or PO. The weight of a node g is written as $w(g)$. A weight of PO is defined as the maximum sequential depth of the PO from all PIs. However, weight of a non-PO node is defined based on whether the node is FF or not and the weight of its fanout nodes. Due to reconvergent fanouts, a unique weight assignment may not be possible without creating a copy of a node if the node has reconvergent fanouts, in which case, the algorithm will make a copy of such a node, as needed, to allow unique weight assignment. For the original node and copied node(s), we assign a permanent tag number called the difference in depth (DID). We use the notation DID(g) to denote the DID of a node g. By default, the DID of a node that has never been duplicated is 0. If weights of the fanout nodes are the same with respect to a PO, a node is *balanced* with respect to that PO. When each node in the circuit is balanced with respect to all POs, the circuit is balanced and can be converted to balanced ATPG model. BAM provides separate origins for unbalanced reconvergent paths. After BAM is generated, we obtain the C-ATPG circuit by replacing FFs in BAM with buffers. The algorithm below contains the exact and formal details of these concepts.

Algorithm 1: Generate a C-ATPG Circuit:

Assign weights to POs: In a single PI-to-PO pass, assign weights to all PO nodes.

BAM generation: Determine node weights separately with respect to each PO. We proceed from a PO toward PIs assigning weights to nodes as discussed in the two cases given below. The weight assigned to a balanced node is the weight of its fanouts reachable from the PO being considered. The weight of a combinational node is the same as

that of a fanout node, and for an FF node, it is one less than that of the fanout node. For each PO, we recursively determine weights of all reachable nodes while eliminating any unbalances. Two cases can occur as a non-PO node g_i is repeatedly processed:

Case 1) Node g_i either has a single fanout node g_j with assigned weight $w(g_j)$, or has multiple fanout nodes, all of which have the same weight $w(g_j)$. Then, $w(g_i) = w(g_j)$, if g_i is a combinational gate or PI, or $w(g_i) = w(g_j) - 1$, if g_i is a FF.

Case 2) Node g_i has fanouts to multiple nodes $g_q, 1 < q \leq n$. Suppose there are $m, m \leq n$, distinctly different weights among these n nodes.

Step 1) **Grouping fanout weights**: Divide n fanouts into m groups in the ascending order of weights, $w_1, w_2, \ldots, w_m$, where w_1 is smallest among all m fanout weights.

Step 2) **Duplication**: If node g_i has never been duplicated, then g_i is duplicated as m nodes, g_{i1} through g_{im}, each of the same type (i.e., PI, gate, or FF) as g_i. If g_i is a PI, then duplication creates $m - 1$ new PIs. Otherwise, inputs to the duplicated nodes are supplied by adding $m - 1$ new fanouts to each fanin node of g_i. DID of a node g_{ik} is determined as $\mathrm{DID}(g_{ik}) = \mathrm{DID}(g_i) + w_k - w_1$, for $k = 1, 2, \ldots, m$. If node g_i has been duplicated while balancing previous POs, make a copy of node g_i if there is no copy of g_i already having $\mathrm{DID} = \mathrm{DID}(g_i) + w_k - w_1$, for $k = 2, 3, \ldots, m$. If node g_i is duplicated l times, where $l \leq m$ and g_i is not a PI node, add $l - 1$ new fanouts to each fanin node of g_i. Each duplicated node is tagged with appropriate DID.

Step 3) **Splitting of fanout**: Split and move the fanouts of $m - 1$ groups corresponding to weights $w_2, w_3, \ldots, w_m$ from g_i to g_{ik}, where $\mathrm{DID}(g_{ik}) = \mathrm{DID}(g_i) + w_k - w_1$ and $k = 2, 3, \ldots, m$.

Time Frame assignment: Each PI node is assigned a number called a *time frame* (TF). TF initially equals the weight of the PI obtained during balancing with respect to the first PO. While balancing with respect to a subsequent PO, TF equals the weight of the PI plus an offset q, where q can be obtained from another PI with known DID and TF by subtracting its weight from the TF.

Elimination of FFs (C-ATPG circuit generation): Once all POs are processed, the model is balanced and is converted to a combinational ATPG (C-ATPG) circuit for test generation by replacing FFs with buffers. ■

In Algorithm 1, the recursion over POs leaves each PI with a unique weight with respect to each PO and a single unique DID with respect to all POs. Case 1 determines the weight of a node with balanced fanouts and Case 2 handles nodes with unbalanced fanouts. For an unbalanced node, the *duplicate and split* (DAS) procedure moves unbalanced fanouts one level backward (toward PIs). Successive applications of DAS eventually move all unbalances to PIs, whose splitting creates a perfectly balanced structure.

In general, the above procedure splits a PI of the sequential circuit into multiple PIs in the balanced model. Although in the worst case, a gate, FF, or PI can be duplicated up to $d_{\max}$ times, a typical node in a real circuit is copied only a couple of times on average. For test sequentialization, we assign an integer time-frame tag (TF) to each PI of balanced model. This tag identifies the time frame in which the PI supplies the input value to the circuit. The TF is permanent and is assigned when a weight is assigned to a PI for the first time during the balancing procedure. We use weights assigned to PIs while balancing the first PO as TFs, then use normalized weights for subsequent TF assignments as follows: For a PI A with unassigned TF, $\mathrm{TF}(A) = w(A)$, while balancing the first PO, or $\mathrm{TF}(A) = W(A) + \mathrm{PI}_{\mathrm{Offset}}$ while balancing a subsequent PO, where $\mathrm{PI}_{\mathrm{Offset}} = \mathrm{TF}(B) - w(B)$ for another PI B with known TF and $w(B)$ for a subsequent PO. In general, TF equals the weight assigned to a PI. However, in some special cases, we may need to compute and use an offset value, obtained from a PI with the assigned TF and weight with respect to the PO being balanced. In other words, TF is equal to the weight of PI when we balance the first PO, or the weight of PI plus the normalizing offset while balancing subsequent POs.

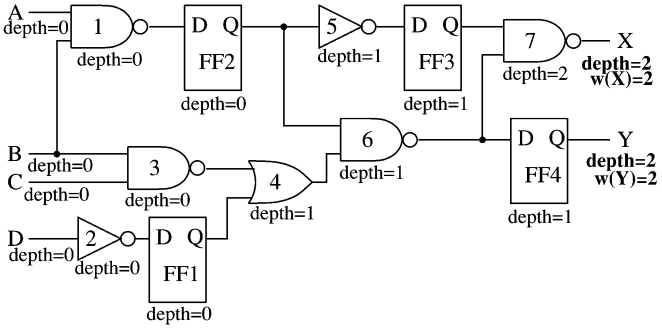

Fig. 4. Assigning weights to primary outputs (POs).

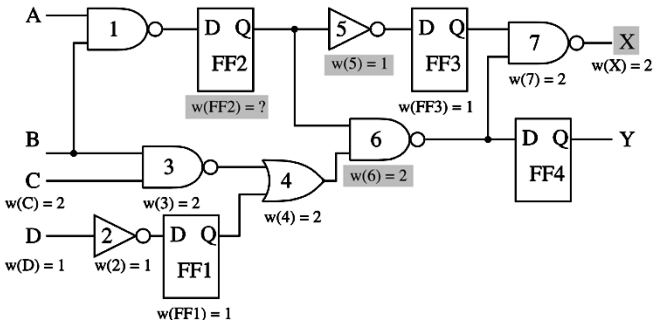

Fig. 5. Conflicting weight assignment on FF2.

Let us generate the BAM for the circuit in Fig. 1. First, we assign the weights to PO nodes, X and Y as shown in Fig. 4. The sequential depth of a node is defined as the maximum number of sequential elements that appear on any path from its fanins to any reachable PIs. For example, FF1 has only one fanin and its sequential depth is 0. Gate 5 has only one fanin and its fanin node is a FF. Thus, we assign 1, the sequential depth of its fanin node, FF2, plus 1. Similarly, gate 4 has two fanins and they have sequential depths of 0 and 1. Since we take the maximum of 0 and 1, we assign 1 as its sequential depth. The depths of all nodes in the circuit can be assigned in linear time, *O(n)*, where n is the number of nodes in the circuit. The weight of a PO node is the sequential depth of the PO so we assign $w(X) = 2$ and $w(Y) = 2$ (shown in boldface in Fig. 4). This completes the first item of Algorithm 1.

We start generating the BAM using the PO weights. According to the algorithm, we balance the circuit with respect to one PO at a time. First, we balance the circuit with respect to PO X. We start with X and recursively assign weights to its fanin nodes. The weight of a node is defined as its fanout node's weight if it is not a FF, or fanout node's weight minus 1 if the node is a FF. Since gate 7 is the only fanin of X and it is not a FF, we assign 2 as its weight, $(w(7) = w(X) = 2)$. We then assign weights to nodes FF3, 5 and 6. As highlighted by shading in Fig. 5, there is a conflict in weight assignment of FF2 while balancing the circuit with respect to PO X. Weights on two fanout nodes of FF2 are $w(5) = 1$ and $w(6) = 2$. Using Algorithm 1, we first place them into two groups, corresponding to weights 1 and 2, respectively, as $\{5\}$ and $\{6\}$. Then, we duplicate the node FF2 and split the fanout as illustrated in Fig. 6(a). Since no copy of FF2 has been made before, $\mathrm{DID}(FF2) = 0$ and we assign $\mathrm{DID} = 0$ to the original node and $\mathrm{DID} = 0 + w(6) - w(5) = 1$ to the copied node. Using

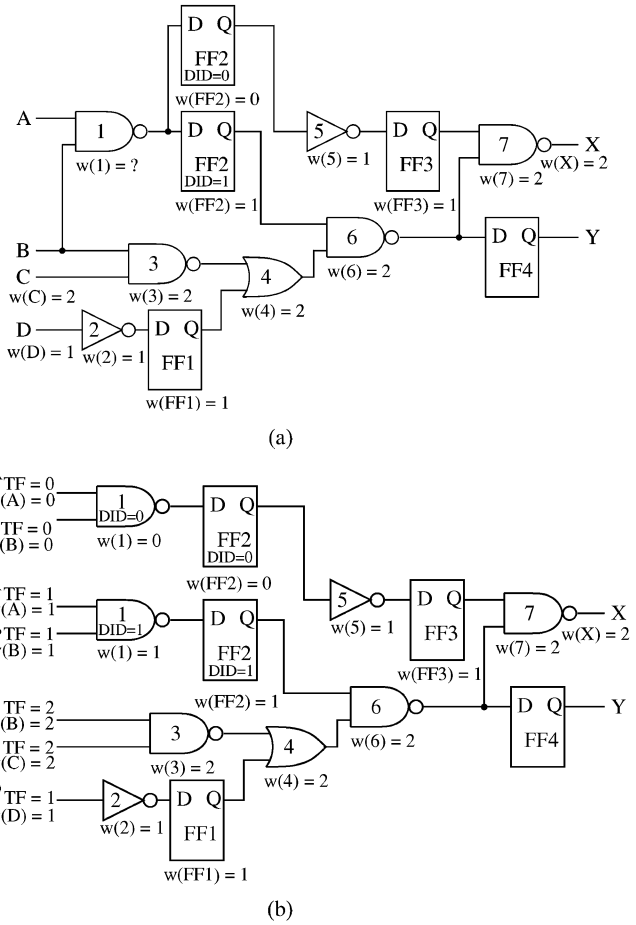

Fig. 6. Duplicate and split (DAS) operations. (a) DAS operation on FF2. (b) DAS operation on Gate 1, PI A, and PI B.

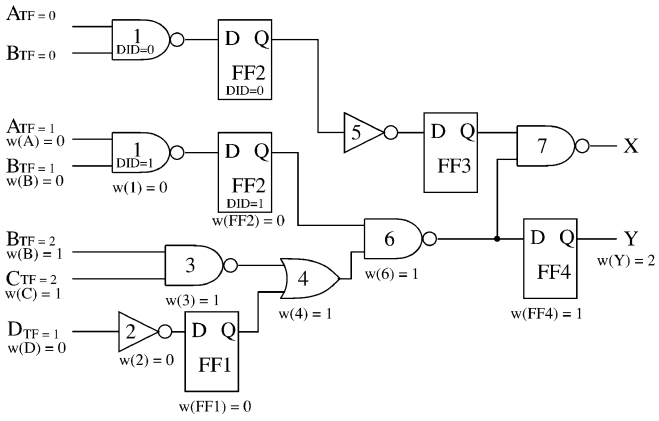

Fig. 7. Balancing with respect to PO Y.

Case 2 of Algorithm 1, the weight of node $\mathrm{FF2}_{\mathrm{DID}=0}$ is $w(5)-1=0$ and weight of $\mathrm{FF2}_{\mathrm{DID}=1}$ is $w(6)-1=1$. We recursively assign the weights to the fanin nodes of FF2 and its copy. Fig. 6(b) shows the result of applying DAS to node 1 and PIs A and B to complete the balancing with respect to X. As we complete the balancing with respect to the first PO, we have assigned weights to all PIs as their TFs.

Next, to balance the circuit with respect to Y, we assign a weight of 1 to node 6, $w(6)=w(\mathrm{FF4})-1=1$. Similarly, weights are assigned to all of its fanin nodes without any conflict as shown in Fig. 7. All PIs have been assigned TFs from balancing previous PO and they are not changed even though they have an offset of 1 with respect to Y.

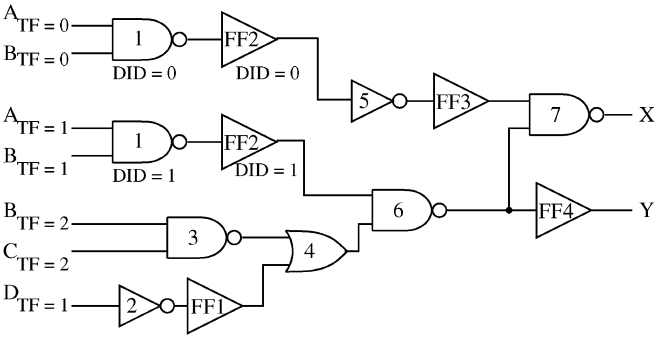

Fig. 8. C-ATPG circuit for the circuit of Fig. 1.

The use of DAS provides separate signal paths in the combinational model so that the copies of a signal may assume different values in different time frames. The recursive DAS transformation may create a circuit with new PIs having specific weights. When all nodes are balanced with respect to both POs, X and Y, we replace all FFs with buffers to obtain a C-ATPG circuit as shown in Fig. 8.

Regardless of the order in which POs are balanced, only one unique balanced model will be generated. However, depending on the order of the POs during balancing, the TF labels in the balanced model may differ by a constant offset. This is due to the fact that TFs are normalized with respect to the first PO. If the circuit of Fig. 1 was balanced with respect to PO Y first, and then X, the resultant C-ATPG circuit will have exactly the same structure; only the (TF) labels at PIs will differ by a constant offset -1.

B. Test Generation

If the DAS transformation has created more than one copy of the fault site in the BAM, then all copied lines and the original line are the sites of simultaneously occurring faults in the BAM modeled as a set of *multiple fault*s, otherwise a fault has a single fault mapping. If there is a test for a *mapped* single fault in the BAM, then the corresponding fault will be detected in the sequential circuit by the sequentialized test. Otherwise, the fault is undetectable. Similarly, if there is a test for a *mapped* multiple fault in the BAM, then the corresponding fault is detectable in the sequential circuit. Otherwise, the fault is undetectable. This follows from the fact that Algorithm 1, which splits the fault site, preserves the circuit output function. Therefore, the original circuit with a single fault and the BAM with the mapped multiple fault are functionally identical. This implies that the single fault is *equivalent* to the mapped multiple fault [4]. As a consequence, the fault masking that is sometimes possible among the components of a multiple fault [4] cannot occur when the single fault in the original circuit is detectable.

Combinational tests generated for BAM must be sequentialized for application to the original sequential circuit. Let $d_{\max}$ denote the maximum sequential depth of the original sequential circuit. As explained earlier, the weights assigned to PIs in the BAM are called *time frames* (TFs). TFs determine the time sequence of signal values at the PI. For example, there are three copies of the PI B of the circuit of Fig. 1 with $\mathrm{TF}=0, 1$, and 2 as shown in Fig. 8, and they represent the PI in time frame 0, 1, and 2, respectively. Using the TF on each PI, we can convert a combinational test vector of BAM to a sequential test. Each combinational vector produces a sequence of vectors of length *up to* $d_{\max}+1$. In general, the vector sequence length can be less than $d_{\max}+1$, since we only need to propagate the effect of the fault to an observable PO. We transform each combinational test vector generated for BAM to a vector sequence for the original circuit as follows:

Algorithm 2: Test-Transformation: Suppose that the BAM has $k+1$ copies of a PI node A with TFs of $\{t_0, t_1, \ldots, t_k\}, k \leq d_{\max}$, sorted in ascending order, and the values on these $k+1$ copies in a combinational

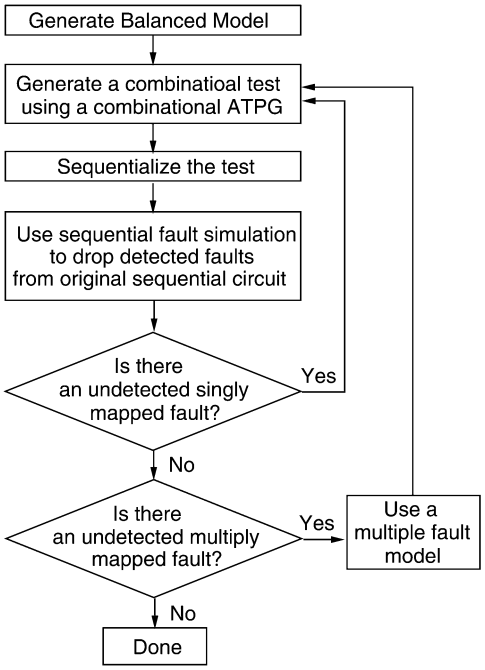

Fig. 9. Test generation with C-ATPG and sequential fault simulation.

test derived for BAM are $\{v_0, v_1, \ldots, v_k\}$. The value of the PI A in the t_ith vector of the sequence is v_i. If some value $t_j, 0 \leq j \leq k$ does not occur among the TFs of A, then the value of the previous time frame is assigned to the PI A in the t_jth vector. Leading vectors with all Xs are deleted, but the total length of the vectors must be $d_{\max} + 1$ to guarantee the detection of the fault. ■

Fig. 9 shows a flowchart of our test generation method. First, we generate BAM. Using a C-ATPG, we generate a test for a single fault in BAM. We then sequentialize the combinational test and fault-simulate the original acyclic circuit using a sequential circuit fault simulator. All detected faults are dropped from the undetected fault list. We repeat the test generation, test sequentialization, and fault simulation steps until there is no more undetected single fault. If there are any undetected multiple faults, we generate a test for each multiple fault using the multiple fault model [22], [24]. Following our previous work [22], [24], Ichihara and Inoue [15] proposed a method of modeling multiple faults by a single fault so that a single-fault C-ATPG can be used for test generation. Their model is similar to the one proposed earlier [22]. Both methods used added logic. The method of [15] requires additional *ground* and V_{cc} signals, whereas that of [22] adds extra PI(s) on which single faults are inserted to model multiple faults. We have also proposed a more elegant solution to this problem in [24], which is suitable for general multiple fault modeling and is described in the following section.

IV. Generalized Model for Multiple Faults

Fig. 10(a) shows four lines with inputs a, b, c, and e, and the respective outputs A, B, C, and E. A multiple stuck-at fault here consists of the first two lines stuck-at-1 and the others stuck-at-0. A multiple-fault involving any set of lines and any arbitrary fault conditions can be modeled in a similar manner. We will use the multiple stuck-at fault example of Fig. 10(a) for explaining the modeling method.

The model in Fig. 10(b) contains a single stuck-at fault. It consists of two types of gates [24].

1) *In-line gates:* A two-input gate is inserted in each faulty line. The controlling input signal state for this gate is the same as the value at which the line is stuck. Thus, an AND (OR) gate is inserted in a line that is stuck-at-0 (1). When the fault on a line is not activated, the in-line gate forces the correct value on it.

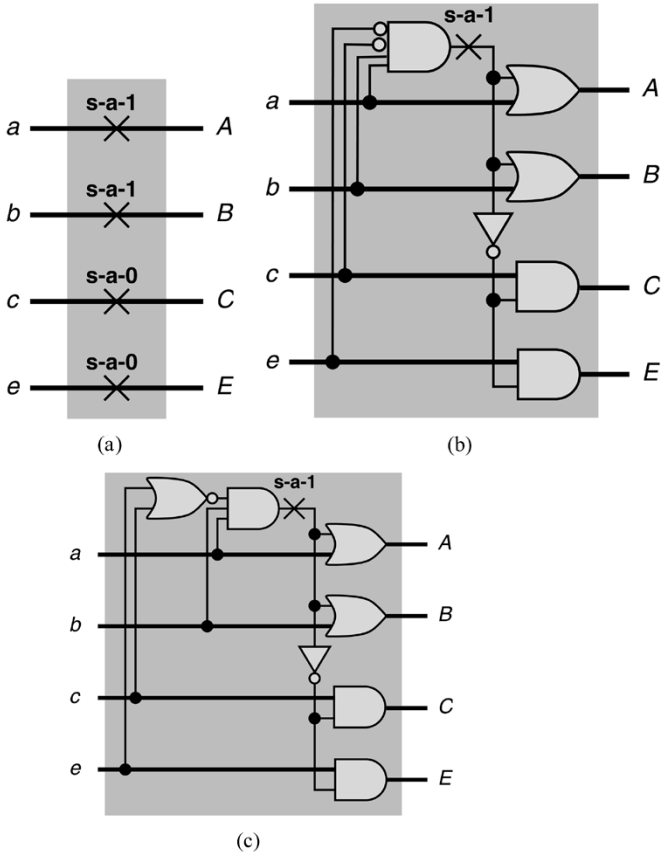

Fig. 10. Model for a multiple stuck-at fault. (a) A multiple stuck-at fault. (b) An equivalent single stuck-at fault. (c) An alternate implementation.

2) *Fault gate:* This is an n input AND gate that feeds all in-line gates either directly if the in-line gate is OR type, or through an inverter if it is AND type. The inputs to the fault gate are derived directly from all s-a-1 lines and via inversions from all s-a-0 lines. A single s-a-1 fault is modeled at the output of this gate. Thus, whenever one or more faults are activated, the single fault in Fig. 10(b) is activated as $\bar{D}$. We use the symbols D and $\bar{D}$ to represent the state of a line affected by fault. $D(\bar{D})$ means that the fault-free (faulty) value is 1 (0) and faulty (fault-free) value is 0 (1) [4]. As a result, in-line gates inject $\bar{D}$ on all lines on which s-a-1 faults are activated and D on lines on which s-a-0 faults are activated. A nine-valued algebra [28] may be required for proper test generation if the circuit was sequential.

We validate the model using the example of Fig. 10. The model would be correct if two conditions hold.

1) **Condition 1:** *Circuit equivalence*. Fault-free output functions must be identical for the original circuit and the model. For Fig. 10(b), we have: $A = a + ab\bar{c}\bar{e} = a, B = b + ab\bar{c}\bar{e} = b, C = c(\overline{ab\bar{c}\bar{e}}) = c(\bar{a} + \bar{b} + c + e) = c + c(\bar{a} + \bar{b} + e) = c, E = e(\overline{ab\bar{c}\bar{e}}) = e(\bar{a} + \bar{b} + c + e) = e + e(\bar{a} + \bar{b} + c) = e$, which are same for the fault-free circuit in Fig. 10(a). ■
2) **Condition 2:** *Fault equivalence*. For the single s-a-1 fault in Fig. 10(b) to be equivalent to the multiple stuck-at fault in Fig. 10(a), each of the four faulty functions should be identical [4]. Faulty functions for multiple-fault (mf) in Fig. 10(a) and those for single-fault (sf) in Fig. 10(b) are: $A_{\rm mf} = 1; A_{\rm sf} = a + 1 = 1, B_{\rm mf} = 1; B_{\rm sf} = b + 1 = 1, C_{\rm mf} = 0; C_{\rm sf} = c.0 = 0, E_{\rm mf} = 0; E_{\rm sf} = e.0 = 0$. Thus, fault equivalence is established. ■

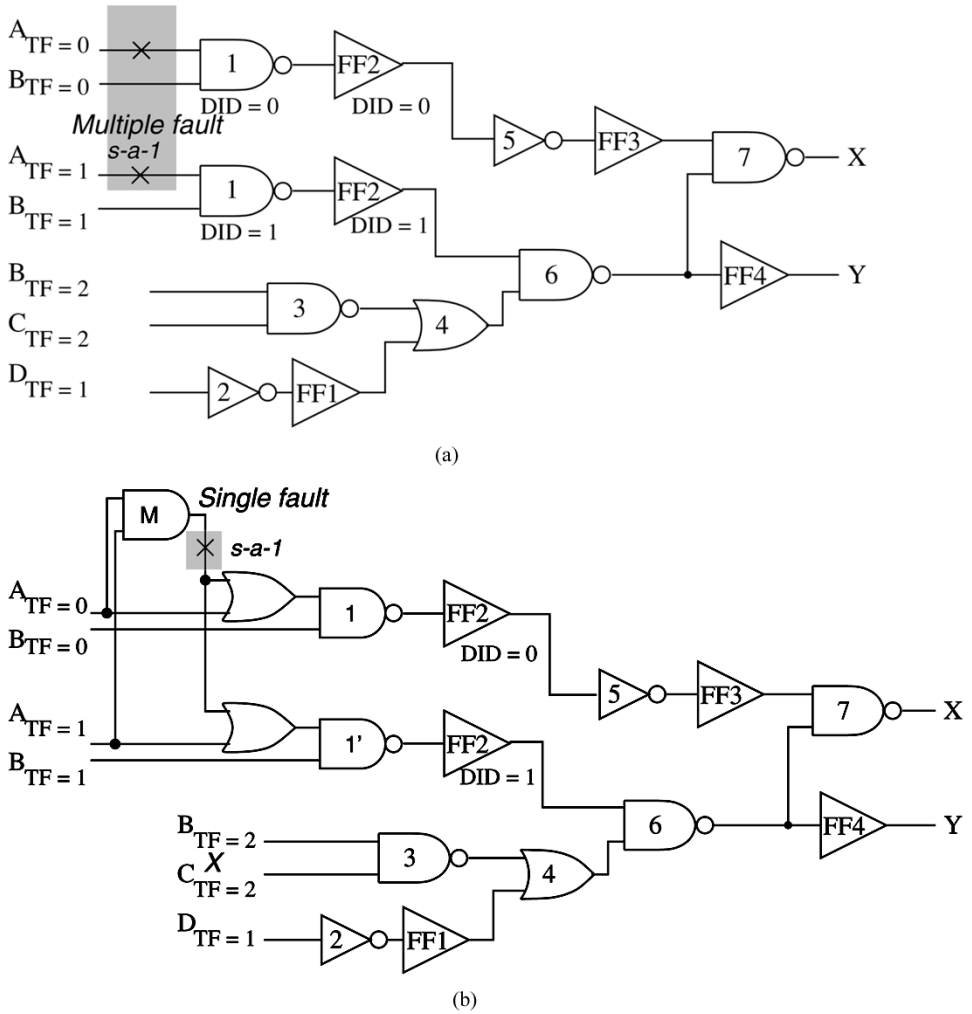

Fig. 11. C-ATPG circuit with a multiple fault. (a) A combinational model with multiple faults $A_{\mathrm{TF}=0}$ and $A_{\mathrm{TF}=1}$. (b) A combinational model with the equivalent single fault M s-a-1.

Although this validation is carried out for a specific example, it is done without any assumptions. Hence, we regard the model to be valid for a multiple fault in any combinational circuit and involving any arbitrary set of single stuck-at faults.

A multiple stuck-at fault of multiplicity n can be modeled as a single stuck-at fault by using at most $n + 3$ gates. Let us assume that out of n faults, there are $(n-k)$ s-a-1 and k s-a-0 faults, where $n \geq k$. Fig. 10(c) shows an alternative implementation using only $n + 3$ additional gates. These include n two-input *in-line* gates, one $(n-k+1)$-input *fault* AND gate, one k-input NOR gate, and one inverter. When all n-component stuck-at faults are of the same type, as is the case in our application, then only $n + 1$ gates will be required.

Fig. 11(a) shows the C-ATPG circuit for the circuit of Fig. 1. The input A stuck-at-1 fault in Fig. 1 appears as a multiple-fault in the C-ATPG circuit in Fig. 11(a). Using the proposed multiple fault modeling method, we add three gates to model the multiple fault $A_{\mathrm{TF}} = 0$ s-a-1 and $A_{\mathrm{TF}} = 1$ s-a-1 by a single stuck-at fault M s-a-1 as shown in Fig. 11(b). As a result, a conventional C-ATPG can be used to generate the test for the multiple fault.

Fig. 9 shows a test generation method using a sequential fault simulator. With the use of the multiple fault model, the sequential fault simulator is not required and fault simulation is possible using the combinational fault simulator. First, we generate BAM and add logic to model all multiple faults as single faults. Using a combinational ATPG, we generate a test for a mapped fault in BAM. We then simulate mapped faults for the generated combinational test applied to the C-ATPG circuit using a combinational fault simulator. All detected faults are dropped from the undetected fault list. We repeat the test generation, test sequentialization, and fault simulation steps until there is no more undetected fault.

V. Experimental Results

The proposed combinational model is applicable to all classes of acyclic sequential circuits. If the given circuit is not acyclic, a partial-scan procedure [1], [5] can be used to make it acyclic. In this section, we consider those ISCAS '89 benchmark circuits that are either acyclic or have at least one FF remaining after making them acyclic via partial scan.

We have implemented the algorithms presented in this paper in a C language program. In our program, a *weighted* directed acyclic graph (DAG) is generated for the circuit and then Algorithm 1 is applied. The vertices of the DAG are PIs, gates, and POs. An arc between a vertex-pair represents a signal flow path and the integer weight of the arc equals the number of FFs on the path.

Table I shows statistics for applicable ISCAS '89 benchmark circuits for our method. Three columns under "Flip-flops (FFs)," namely "Total," "Scan," and "Scan (%)," give the total number of FFs, the number of scan FFs used to make the circuit acyclic, and the percentage of scan FFs, respectively. The column "Max. Depth" shows the max-

TABLE I
CIRCUIT STATISTICS

Circuit Name	Flip-flops (FFs)			Max. Depth	Number of		
	Total	Scan	Scan (%)		PIs	POs	Gates
s382	21	15	71.4	1	18	21	164
s400	21	15	71.4	1	18	21	168
s444	21	15	71.4	1	18	21	187
s641	19	15	78.9	1	50	39	383
s713	19	15	78.9	1	50	38	397
s953	29	6	20.7	1	22	29	418
s1196	18	0	0	3	14	14	547
s1238	18	0	0	3	14	14	526
s1423	74	71	95.9	1	88	76	660
s5378	179	30	16.8	19	65	79	2928
s9234	228	152	66.7	4	171	174	5673
s13207	669	310	46.3	22	341	431	8310
s15850	597	438	73.4	29	455	528	9928
s35932	1728	306	17.7	34	341	626	17487
s38417	1636	1080	66.0	9	1108	1186	22735
s38584	1452	1115	76.8	35	1127	1393	19590
Average	421	226	53.4	10	244	293	5631

imum sequential depth of the *acyclic* circuits after removing the scan FFs. The last three columns give the numbers of PIs, POs, and gates after converting each scan FF to a PI-PO pair. On average, about 53% of FFs are scanned to make the circuit acyclic.

The C-ATPG model has overhead associated with it as it increases the size of the circuit for which the test generation should be performed. Table II shows detailed statistics for generated models of ISCAS '89 benchmark circuits. The second and third column of this table give numbers of PIs and gates in the C-ATPG circuits. The numbers of POs of C-ATPG circuits are identical to those of the corresponding acyclic sequential circuits since we do not duplicate POs. The next two columns show ratios of *PIs* and gates for the C-ATPG circuit over the acyclic sequential circuit. The C-ATPG circuits have about 166% more PIs (2.66 in ratio) and 59% more gates (1.59 in ratio) over the acyclic sequential circuits. Another useful measure is comparing the number of nodes, where nodes are PIs, gates, and POs. The next, three columns under "No. of nodes (PIs+gates+POs)" give the number of nodes of acyclic sequential circuits (Acy.), C-ATPG circuits (C-ATPG), and the ratio of nodes (ratio). The C-ATPG circuit has about 97% (1.97 in ratio) modeling overhead on an average compared to the acyclic sequential circuit.

The next three columns under "No. of Faults" show the numbers of total faults (Total), singly-mapped faults (Single) and multiply-mapped faults (Mult.) in the C-ATPG model. The column under "Avg. Multi." shows the average multiplicity of multiply mapped faults in each circuit and "% of MF" shows the percent of multiply-mapped faults out of total faults. We note that the multiplicity of these faults is highly correlated to the maximum depth of the circuit. If the maximum depth of the circuit is large, average multiplicity of the fault is also large. On average, about 25% of faults are multiply mapped.

The balanced ATPG model requires the single-to-multiple fault mapping because of the splitting of signals. These multiple faults correspond to faults at the same site in the original circuit at different timeframes. Only in cases where such multiple faults mask each other is a multiple fault model required for test generation. When the C-TPG program is not equipped with multiple fault detection capability, we found that a single fault assumption still produces tests for most faults. However, we can easily model the multiple faults with a simple modification as described in Section IV. In general, the test generation time for such multiply-mapped faults is similar to that of single faults.

As shown in Table II, approximately 25% of faults are multiply-mapped faults and we need to handle them correctly. If we were to add the logic to model all the multiply-mapped faults as single faults in the C-ATPG circuits, the average model overhead is 220% and as high as 699% for one benchmark circuit [21]. In spite of the added model overhead, the C-ATPG method can speed up the test generation 6.3 times on an average over the conventional sequential test generation approach for ISCAS '89 benchmark circuits [21]. For the experiment, the GENTEST [6] ATPG is used for sequential test generation on sequential circuits and the TetraMax [30] is used for combinational test generation on C-ATPG circuits on a 440-MHz Sun Ultra 10 workstation with 512 MB of memory. Performance of another sequential ATPG, HITEC [29], was found to be similar and within $\pm 10\%$ of GENTEST. Better improvement is possible and one of the methods is described below.

The purpose of our C-ATPG method is to reduce test generation time using a more efficient combinational ATPG. We employ a two-pass strategy first using the C-ATPG circuit without multiply-mapped faults and then using a modified C-ATPG circuit with *selected* multiply-mapped faults modeled as single faults. As shown in Fig. 9, we first target singly-mapped faults. Then, we simulate all faults. Since fault simulation is fairly efficient for both combinational and sequential circuits, we can either simulate the acyclic sequential circuit on a sequential fault simulator or simulate the combinational ATPG circuit with added logic for all multiple faults using a combinational fault simulator. After dropping all detected single and multiple faults, we can model only those multiple faults that are not detected by the test vectors of single faults. Even though this method has the disadvantage of reading and preprocessing two different circuit descriptions, total ATPG times were significantly faster than modeling all multiple faults as single faults.

Table III shows the results of the above experiment using TetraMax and GENTEST. The two columns under "TG stat." show the fault coverages (FC) and fault efficiencies (FE) for both combinational and sequential methods. Next, the three columns under "Combinational ATPG" show the combinational ATPG CPU times for C-ATPG circuits. Column "SF ATPG (s)" shows the ATPG time in seconds for unmodified C-ATPG circuits to target all singly-mapped faults. During fault simulation, all detected multiply-mapped faults are dropped from the fault list. Then, we modify the C-ATPG circuit so that all undetected multiply-mapped faults are modeled as single faults. The column "MF ATPG (s)" shows the ATPG time in seconds for targeting the remaining undetected multiple faults. The next column under "Total ATPG (s)" shows the test generation time in seconds obtained as the sum $SF\ ATPG + MF\ ATPG$. The sequentialized vector length "(Seq. VL)" and the percentage of multiple faults "% of MF" that required single-fault modeling appear in the next two columns. Then, follow two columns under "Sequential ATPG" that give the total ATPG CPU time in seconds (Total ATPG (s)) and sequential vector length (Seq. VL) of sequential ATPG. No efforts were made to compact the vector lengths, which can be reduced via various combinational and sequential compaction algorithms and methods.

In *all* cases, the combinational ATPG method yielded equal or better fault coverages (FC) and fault efficiencies (FE). The combinational ATPG method gave a higher fault efficiency for s9234, as two aborted faults during sequential ATPG were also detected. Higher fault coverage (one more fault detected) was obtained for s38584 in less time than the sequential ATPG.

The last column "Seq/Comb ATPG time ratio" shows the ratio of total ATPG time of sequential ATPG over that of the combinational ATPG method. This is the speed up of the new combinational ATPG approach over the conventional sequential ATPG method, which, on average, is 17.4 times. In comparing the results from two different ATPG programs, we must make sure that we choose the right parameters and results for comparison. Some ATPGs may spend more time in

TABLE II
STATISTICS FOR C-ATPG CIRCUITS OF PARTIAL-SCAN ISCAS'89 CIRCUITS

Circuit name	No. of PIs	No. of gates	PI ratio	gate ratio	No. of nodes (PIs+gates+POs) Acy.	C-ATPG	ratio	No. of Faults Total	Single	Mult.	Avg. Multi.	% of MF
s382	18	164	1.00	1.00	203	203	1.00	399	399	0	N/A	0.0
s400	18	168	1.00	1.00	207	207	1.00	424	424	0	N/A	0.0
s444	18	187	1.00	1.00	226	226	1.00	474	474	0	N/A	0.0
s641	51	385	1.02	1.01	472	475	1.01	467	465	2	2.00	0.4
s713	51	399	1.02	1.01	485	488	1.01	581	579	2	2.00	0.3
s953	22	418	1.00	1.00	469	469	1.00	1079	1079	0	N/A	0.0
s1196	49	815	3.50	1.49	575	878	1.53	1242	770	472	2.69	61.3
s1238	49	792	3.51	1.51	554	855	1.54	1355	853	502	2.66	58.9
s1423	94	665	1.07	1.01	824	838	1.00	1515	1499	16	2.00	1.1
s5378	423	9426	6.50	3.22	3072	10023	3.26	4603	3133	1470	8.25	46.9
s9234	367	7523	2.14	1.33	6018	8321	1.38	6514	4974	1540	2.72	31.0
s13207	1133	17924	3.32	2.16	9082	19133	2.11	9815	6426	3389	4.63	52.7
s15850	3177	33765	6.98	3.40	10911	37470	3.43	11725	7084	4681	9.43	66.1
s35932	1296	31238	3.80	1.79	18454	33160	1.80	39094	28336	10758	4.31	38.0
s38417	1819	25862	1.64	1.14	25029	28867	1.15	31180	27770	3410	2.53	12.3
s38584	4649	47240	4.13	2.41	22110	53282	2.41	36303	27743	8560	6.88	30.9
Average	827	11061	2.66	1.59	6168	12181	1.97	9173	7001	2175	4.18	25.0

TABLE III
TEST GENERATION RESULTS

Circuit name	T.G. Stat. FC (%)	FE (%)	Combinational ATPG (Figure 9) SF ATPG (s)	MF ATPG (s)	Total ATPG (s)	Seq. VL	% of MF	Sequential ATPG Total ATPG (s)	Seq. VL	Seq/Comb ATPG time ratio
s382*	100.0	100.0	0.04	0.00	0.04	86	0.00	0.07	80	1.7
s400*	98.6	100.0	0.03	0.00	0.03	89	0.00	0.07	75	2.2
s444*	97.0	100.0	0.05	0.00	0.05	78	0.00	0.10	83	2.0
s641	100.0	100.0	0.06	0.00	0.06	123	0.00	0.15	116	2.5
s713	93.5	100.0	0.09	0.00	0.09	126	0.00	0.40	117	4.4
s953*	100.0	100.0	0.13	0.00	0.13	190	0.00	0.45	168	3.5
s1196	99.8	100.0	0.32	0.00	0.32	382	0.00	1.08	323	3.4
s1238	94.7	100.0	0.33	0.18	0.51	400	0.60	1.72	317	3.4
s1423	99.1	100.0	0.13	0.00	0.13	182	0.00	1.20	185	9.2
s5378	93.7	99.8	6.61	8.12	14.73	1232	2.04	925.87	1117	62.9
s9234	93.2	99.9	2.69	6.69	9.38	1681	4.22	323.95	1300	34.5
s13207	97.1	100.0	5.78	8.52	14.30	2965	1.45	238.92	2442	16.7
s15850	96.7	100.0	11.11	16.60	27.71	3925	1.37	457.53	2620	16.5
s35932	89.8	100.0	8.85	10.60	19.45	6564	1.26	917.62	2377	47.2
s38417	99.3	100.0	12.70	2.52	23.22	7254	1.09	803.43	5360	34.6
s38584	95.7	100.0	16.10	31.94	48.04	9720	0.37	1596.60	5763	33.2
Average	96.8	100.0	4.06	5.82	9.89	2187	1.03	289.7	1402	17.4

"learning" for faster test generation, while others may spend less time in learning and may be penalized in test generation time. To be fair, we report the total ATPG time, which includes time required for reading netlists, design rule checking, learning, test generation, and fault simulation. Although not reported here when several sequential ATPG programs, such as GENTEST [6], HITEC [29], and FASTEST [20], were used on both acyclic sequential circuits and C-ATPG circuits, our new combinational method produced equal or better coverages with an average 40% less CPU time [21] despite the increased PIs and gates in the C-ATPG circuits.

VI. CONCLUSION

The new test generation method using a combinational model for a general acyclic sequential circuit requires only a combinational ATPG to achieve equal or better fault coverage and fault efficiency than a conventional sequential ATPG. Our test generation method is based on transforming the unbalanced acyclic sequential circuit to a combinational model by moving all unbalanced fanouts to PIs and then adding new PIs. Added PIs allow combinational model to create nonrepeated vectors to detect a fault in the original acyclic sequential circuit. Because a combinational model is used to generate tests and the test generation time spent on detectable as well as untestable faults is significantly lower, we can expect a 100% fault efficiency. However, a drawback of the the proposed method is that the increases in the number of PIs and gates in the test generation model may increase the learning time, which in tern may negate the benefits of faster test generation with 100% fault efficiency. Thus, the advantage should be weighed against the increased size of the C-ATPG circuit that the test generation program must deal with.

The multiple-fault model we present allows the C-ATPG circuit to be used with any C-ATPG. Our motivation is to make the C-ATPG method applicable using the available single-fault ATPG tools. Extremely efficient single-fault ATPG programs are commercially available [30] for which increases in the number of gates do not present any difficulty. We believe this will lead to future research on novel design and test methods. Several other applications of the multiple fault mapping technique, such as, diagnosis, circuit optimization, and detection of bridging and multiple faults, have been discussed in our recent work [21], [24].

ACKNOWLEDGMENT

The authors thank A. Balakrishnan for supplying the MFVS for several ISCAS'89 circuits. The views expressed in this paper are those of the authors and do not reflect the official policy of or position of the U.S. Air Force, the Department of Defense, or the U.S. Government.

REFERENCES

[1] "Special issue on partial scan methods," *J. Electronic Testing: Theory and Applic.*, vol. 7, no. 1/2, Aug./Oct. 1995.
[2] V. D. Agrawal, K.-T. Cheng, D. D. Johnson, and T. Lin, "Designing circuits with partial scan," *IEEE Design Test Comput.*, vol. 6, no. 2, pp. 8–15, Apr. 1988.
[3] A. Balakrishnan and S. T. Chakradhar, "Sequential circuits with combinational test generation complexity," in *Proc. 9th Int. Conf. VLSI Design*, Jan. 1996, pp. 111–117.
[4] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. Norwell, MA: Kluwer, 2000.
[5] K.-T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 544–548, Apr. 1990.
[6] W. T. Cheng and T. J. Chakraborty, "GENTEST: An automatic test generation system for sequential circuits," *Computer*, vol. 22, no. 4, pp. 43–49, Apr. 1989.
[7] D. L. Dietmeyer, *Logic Design of Digital Systems*, 3rd ed. Boston, MA: Allyn and Bacon, 1988.
[8] E. B. Eichelberger, E. Lindbloom, J. A. Waicukauski, and T. W. Williams, *Structured Logic Testing*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[9] H. Fujiwara, "A new class of sequential circuits with combinational test generation complexity," *IEEE Trans. Comput.*, vol. 49, no. 9, pp. 895–905, Sep. 2000.
[10] ——, "A new definition and a new class of sequential circuits with combinational test generation complexity," in *Proc. Int. Conf. VLSI Design*, Jan. 2000, pp. 288–293.
[11] R. Gupta and M. A. Breuer, "Testability properties of acyclic structures and applications to partial scan design," in *Proc. IEEE VLSI Test Symp.*, Apr. 1992, pp. 49–54.
[12] ——, "Partial scan design of register-transfer level circuits," *J. Electron. Testing: Theory Applicat.*, vol. 7, no. 1/2, pp. 25–46, 1995.
[13] R. Gupta, R. Gupta, and M. A. Breuer, "The BALLAST methodology for structured partial scan design," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 538–548, Apr. 1990.
[14] D. A. Huffman, "Combinational circuits with feedback," in *Recent Developments in Switching Theory*, A. Mukhopadhyay, Ed. New York: Academic, 1971.
[15] H. Ichihara and T. Inoue, "Test generation for acyclic sequential circuits with single stuck-at fault combinational ATPG," in *Proc. Design, Automation, Test Eur. Conf. Exhib.*, Mar. 2003, pp. 1180–1181.
[16] M. Inoue, E. Gizdarski, and H. Fujiwara, "Theorems for separable primary input faults in internally balanced structures," *Inform. Sci.*, no. , pp. 1–5, Mar. 2000.
[17] ——, "Sequential circuits with combinational test generation complexity under single-fault assumption," *J. Electron. Testing: Theory Applicat.*, vol. 18, no. 1, pp. 55–62, 2002.
[18] T. Inoue, D. K. Das, T. Mihara, and H. Fujiwara, "Test generation for acyclic sequential circuits with hold registers," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 550–556.
[19] T. Inoue, T. Hosokawa, T. Mihara, and H. Fujiwara, "An optimal time expansion model based on combinational ATPG for RT level circuits," in *Proc. 7th Asian Test Symp.*, Sep. 1998, pp. 190–197.
[20] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An efficient algorithm for sequential circuit test generation," *IEEE Trans. Comput.*, vol. 42, no. 11, pp. 1361–1371, Nov. 1993.
[21] Y. C. Kim, "Combinational test generation for sequential circuits," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Wisconsin, Madison, Dec. 2002.
[22] Y. C. Kim, V. D. Agrawal, and K. K. Saluja, "Combinational test generation for acyclic sequential circuits using a balanced ATPG model," in *Proc. 14th Int. Conf. VLSI Design*, Jan. 2001, pp. 143–148.
[23] ——, "Combinational test generation for various classes of acyclic sequential circuits," in *Proc. Int. Test Conf.*, Oct. 2001, pp. 1078–1087.
[24] ——, "Multiple faults: Modeling, simulation, and test," in *Proc. 15th Int. Conf. VLSI Design*, Jan. 2002, pp. 592–597.
[25] A. Kunzmann and H. J. Wunderlich, "An analytical approach to the partial scan problem," *J. Electron. Testing: Theory Applicat.*, vol. 1, no. 2, pp. 163–174, 1990.
[26] A. Miczo, *Digital Logic Testing and Simulation*. New York: Harper & Row, 1990.
[27] H. B. Min and W. A. Rog, "A test methodology for finite state machines using partial scan design," *J. Electron. Testing: Theory Applicat.*, vol. 3, no. 2, pp. 127–138, 1992.
[28] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. Comput.*, vol. C-25, no. 6, pp. 630–636, Jun. 1976.
[29] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. Eur. Test Conf.*, Feb. 1991, pp. 214–218.
[30] Synopsys, Inc., TetraMAX ATPG User Guide, Nov. 2000. v2000.11 edition, Doc. Order No. 37043-000 TBD.
[31] T. Takasaki, T. Inoue, and H. Fujiwara, "Partial scan design methods based on internally balanced structure," in *Proc. Asia South Pacific Design Automation Conf.*, Feb. 1998, pp. 211–216.

Dynamically Partitioned Test Scheduling With Adaptive TAM Configuration for Power-Constrained SoC Testing

Dan Zhao and Shambhu Upadhyaya

***Abstract*—Given a system-on-chip with a set of cores and a set of test resources, and the constraints on the total power consumption during test and the maximum width on the top-level test access mechanism (TAM), it is required to optimize overall testing time of the system. To solve this problem, we first generate a power-constrained test compatibility graph and then construct a set of power-constrained concurrent test sets (PCTSs) to facilitate concurrent testing. We then handle the constrained scheduling by adaptively assigning the cores in parallel to the TAMs with variable width and efficiently utilizing the TAM bandwidth such that the tests in the same PCTS have their lengths close to each other. We concurrently schedule the test sets by dynamically partitioning and allocating the tests, and consequently constructing and updating a set of dynamically partitioned PCTSs. This reduces the test cost in terms of overall test time. Simulation study shows the productivity gained by using our integrated scheduling approach.**

***Index Terms*—Adaptive test access mechanism (TAM) configuration, dynamic test partitioning, power constraint, system-on-chip (SoC) test, test compatibility.**

I. INTRODUCTION

Systems-on-chip (SoCs) are designed by integrating intellectual property (IP) cores onto a single chip. As cores are deeply embedded in the SoC, direct access to the cores is usually impossible. Thus, an efficient test access architecture is needed to access the cores, which includes three major components, test source and sink, test

Manuscript received December 12, 2003; revised March 22, 2004 and June 21, 2004. This work was supported in part by a NYSTAR grant from the Microelectronics Design Center, University of Rochester. A preliminary version of this paper was presented at the IEEE VLSI Test Symposium, Napa Valley, CA, April 2003. This paper was recommended by Associate Editor K. Chakrabarty.

D. Zhao is with the Center for Advanced Computer Studies, University of Louisiana, Lafayette, LA 70504-4330 USA (e-mail: dzhao@cacs.louisiana.edu).

S. Upadhyaya is with the Department of Computer Science and Engineering, State University of New York, Buffalo, NY 14260-2000 USA.

Digital Object Identifier 10.1109/TCAD.2005.847893