Short Papers

Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits

Vishwani D. Agrawal and Srimat T. Chakradhar

Abstract— **We give two theorems for identifying untestable faults in sequential circuits. The first, the single-fault theorem, states that if a single fault in a combinational array is untestable then that fault is untestable in the sequential circuit. The array replicates the combinational logic and can have any finite length. We assume that the present state inputs of the left-most block are completely controllable. The next state outputs of the right-most block are considered observable. A combinational test pattern generator determines the detectability of single faults in the right-most block. The second theorem, called the multifault theorem, uses the array model with a multifault consisting of a single fault in every block. The theorem states that an untestable multifault in the array corresponds to an untestable single fault in the sequential circuit. For the array with a single block both theorems identify combinational redundancies. Experiments on ISCAS benchmarks show that using a small array size (typically, two to four blocks) we can identify a large number of sequentially untestable faults.**

I. Introduction

If untestable faults are identified, they can be simply ignored to save on testing effort. Untestable faults are those faults for which no test can be found by a test generation algorithm. In combinational circuits, these faults are redundant and can be removed to reduce hardware. For sequential circuits, the relationship between the untestable and redundant faults is not simple. However, redundant faults form a subset of untestable faults.

There are several reasons why a simple method of identifying sequentially untestable faults is useful. First, highly complex sequential circuit test generation programs waste computing resources while attempting to generate tests for untestable faults. Second, we may try to test the sequentially untestable faults that are combinationally testable by scan-like methods. Third, some untestable faults, that can be further classified as redundant, can be removed from the circuit to reduce hardware.

The problem of identifying untestable faults in a sequential circuit is very complex. Known solutions rely on sequential automatic test pattern generation (ATPG) [8]. In the present work, our objective is to find methods based on combinational ATPG that is known to be less complex. We derive two methods where test generation is performed for combinational circuits of finite size. One method requires test generation for single faults while the other requires detection of multiple faults. Considering the state of the art in ATPG, the first method is more practical. Another advantage of our method is that, unlike some of the published results (see Theorem 1 of Ma *et al.* [17]), we do not enforce a *reset state*. Since we first presented the theorems given in this paper at the *European Test Conference* [3], corrections and extensions [23], and possible applications [14] have started appearing in the literature.

Manuscript received May 15, 1994; revised March 14, 1995 and May 15, 1995. This paper was recommended by Associate Editor K.-T. Cheng.

V. D. Agrawal is with AT&T Bell Laboratories, Murray Hill, NJ 07974 USA.

Srimat T. Chakradhar is with C & C Research Laboratory, NEC, Princeton, NJ 08540 USA.

IEEE Log Number 9413225.

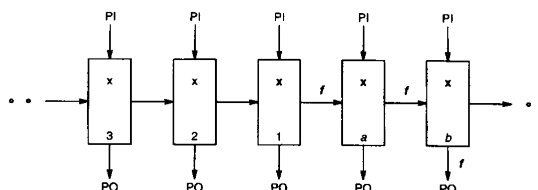

Fig. 1. Combinational array model.

II. Background

Consider the combinational array model of a sequential circuit as shown in Fig. 1. The left input of a block is the present state and the right output is the next state. The input at the top is primary input (PI) and the output at the bottom is primary output (PO). An arrow in the figure represents a group of signals. In sequential circuit test generation, the fault under consideration is assumed to be present in every block. The present state input of the left-most block is assumed to be in an *unknown* logic state. Although a specific reset state is not essential, if one is provided, it must be entirely controllable by PI. The test generator finds input vectors that, when applied to PI, will produce the fault effect at PO.

The fault effect in Fig. 1 is shown as f. The block in which the fault effect first appears is marked as 1. The blocks on the left are 2, 3, etc. The blocks on the right are marked as a, b, etc. Notice that the fault (marked as x) is present in each block but affects the outputs of only blocks 1, a and b. We show a boldface **x** in these blocks to indicate that the fault is active at their outputs. The test is considered complete only when f appears at PO (block b in Fig. 1). In general, the number of blocks on either side of the block 1 can be, if not infinite, large.

III. Theorems

A fault in a sequential circuit can be untestable for various reasons. First, the fault may not be *activated*. In the present discussion, fault *activation* refers to the appearance of fault effect at the boundary of the combinational logic block. Notice that in Fig. 1, only a restricted set of present state inputs can be applied to block 1. Thus, even if the fault is combinationally testable in block 1, block 2 may not produce the state required for fault activation. The second reason for nondetectability of the fault is that it may not be possible to propagate the error to a PO once it has been activated.

A fault in a circuit is called *untestable* if no test can be found with some given test methodology. In the present discussion, we assume that the test methodology corresponds to a gate-level ATPG using a complete branch and bound algorithm. For a testable fault, the test methodology finds a sequence of vectors for which the good and faulty machines produce different logic values at a primary output on the same vector. The discrepancy in the output responses is observed starting from unknown initial states of the two machines. All other faults are classified as untestable. For a detailed classification of untestable faults the reader is referred to a recent paper [22].

A fault in a circuit is called *redundant* if its presence does not alter the input-output behavior of the circuit. Redundant faults are a subset of untestable faults. The input-output behavior of a sequential

0278-0070/95$04.00 © 1995 IEEE

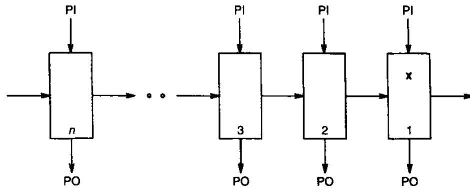

Fig. 2. $C(n)$ with a target single-fault.

circuit can be interpreted in several ways. One may observe the output response immediately after the circuit is powered-up [19]. Alternatively, as is the case in a majority of practical circuits, the output response is observed only after the application of a predetermined sequence of initialization vectors. This sequence takes the machine to a known *initial state* starting from any power-up state. In this case, we will only observe the output response of the machine corresponding to states that are reachable from the initial state. The input-output behavior of a combinational circuit, on the other hand, has one standard interpretation since the output response of the circuit depends only on primary input values. Any set of values that appear on primary inputs at power-up can also be applied to the circuit as an input vector.

For a combinational circuit with only Boolean gates, untestable and redundant faults are identical [1]. For circuits with non-Boolean primitives, like buses, bidirectional devices and tristate gates, not all untestable faults are redundant [7]. In this paper, we only consider circuits with Boolean gates. In a sequential circuit, some untestable faults can be detected by multiple observations [21]. These faults normally interfere with the initialization of the circuit [2]. Many other untestable faults may not be detectable even by multiple observation. Some of these are classified as sequentially redundant. Recent work addresses redundancy in sequential circuits [8], [10], [12], [23]. In this paper, we develop simple techniques to identify untestable faults.

Definition: We define $C(n)$ to be a combinational circuit consisting of blocks $1, 2, \cdots n$. Blocks are arrayed in ascending order from the right to left. Inputs of $C(n)$ are the present state inputs of block n and the primary inputs of all n blocks. Outputs of $C(n)$ are the next state outputs of block 1 and the primary outputs of all n blocks.

Definition: Target single-faults are all single stuck faults in block 1. Since block 1 is a copy of the combinational logic of the sequential circuit, any fault in the sequential circuit, including the input and output faults of flip-flops but excluding the internal faults of flip-flops, is included among the target single-faults.

Fig. 2 shows $C(n)$ with a target single-fault.

Theorem 1 (Single-Fault Theorem): A target single-fault that is untestable in $C(n)$ is also untestable in the sequential circuit.

The proof of this theorem, which appeared in our original paper [3], was analyzed and corrected by Pomeranz and Reddy in a subsequent paper [23]. An interested reader should refer to those papers. Here we give an intuitive discussion on Theorem 1.

Suppose $T(1)$ is the set of all test vectors that detect a target single-fault in $C(1)$. Since $C(1)$ consists of a single copy of the combinational logic of the sequential circuit, each test in $T(1)$ is a single vector. A test vector is specified as a set of values for primary inputs and present state variables of the sequential circuit. Considering all present states required by vectors in $T(1)$ as the *activation states* $A(1)$, we have two cases:

Case 1: set $T(1)$ is empty. In this case, no test is possible for any present state input of block 1. Therefore, the state diagram of the sequential circuit has no activation states. The good and faulty circuits have the same state diagrams. This fault is combinationally redundant.

Case 2: set $T(1)$ is not empty. In this case, there exist activation states $A(1)$ for block 1. Blocks $2, 3, \cdots, n$ can only restrict the possible present state inputs to block 1. If the fault is untestable in $C(n)$, then none of the states in $A(1)$ appear as present state inputs to block 1. If we consider the state diagram, then no state in $A(1)$ can be reached from any state in the state diagram using $n-1$ state transitions. Clearly, no state in $A(1)$ has a self-loop. Otherwise, starting from this state, we can stay in the same state after $n-1$ (in fact, an arbitrary number of) state transitions. Similarly, no state in $A(1)$ can be involved in any cycle. If there is a state S in $A(1)$ that is part of a cycle, then we can always start at a suitable state (possibly S) in the cycle, traverse $n-1$ state transitions in the cycle (one may have to go around the cycle more than once), and reach state S. This is possible for any arbitrary value of n.

If $A(2)$ is the set of immediate predecessors of states in set $A(1)$, then no state in $A(2)$ can be reached from any state in the state diagram using $n-2$ state transitions. If this were possible, then some state in $A(1)$ can be reached in $n-1$ state transitions. Therefore, no state in $A(2)$ can have a self-loop or be on a cycle. In general, if $A(i+1)$ is the set of immediate predecessors of states in set $A(i)$ $(0 < i < n-1)$, then no state in $A(i+1)$ can be reached from any state in the state diagram using $n-i-1$ state transitions. Therefore, no state in $A(i+1)$ can have a self-loop or be on any cycle in the state diagram. This establishes that none of the states in $A(i)$, $0 < i < n-1$, is reachable from a state with self-loop or from a state on a cycle. Note that states in $A(i)$ may have no predecessors. In this case, states in $A(i)$ are not reachable from any other state.

If the initial state for both the good and faulty sequential circuits is a state that is on a cycle in the fault-free state diagram, then the output responses will be identical for both circuits. This is because no activation state in $A(1)$ is reachable from a state on a cycle. For this restricted set of initial states, the good and faulty circuits produce identical output responses for all input sequences. Therefore, the fault, not being detectable from arbitrary initial states, is untestable. Note that the state diagram of a gate-level sequential circuit will always have a cycle since it is a *complete* state diagram.

Theorem 1 relies on checking for the impossibility of fault activation from certain initial states which is a *necessary* condition for detection. The finite size, n, of arrays simply means that we consider a case that is less restrictive than the actual sequential circuit whose valid states may be a subset of all possible states that can be applied to block 1 in $C(n)$.

An understanding of the preceding remark requires a precise notion of the fault-free operation of the sequential circuit. If the circuit has q flip-flops, then it may power up in any one of the 2^q states. If the circuit has a hardware *reset* input, then it can be set to a predetermined initial state. In actual operation, the primary outputs of the circuit are considered irrelevant prior to application of the reset signal. After initialization, the circuit assumes valid states on application of primary input vectors. In general, there are 2^q possible states. However, the valid states are only those reachable from the reset state.

If the circuit has no hardware reset, again, the present state immediately after power up is unknown. The circuit is brought to a known state by applying an initialization input sequence. Also, the primary outputs of the circuit, prior to initialization may be irrelevant. In general, there can be any number of initialization sequences and the corresponding initial states. During fault-free operation, the circuit only assumes states that are reachable from the initial states. Thus, the circuit may assume only a subset of the 2^q states.

Theorem 1 provides a constructive method for identifying untestable faults in sequential circuits that may or may not have a hardware reset state. In circuits with hardware reset, the initialization

of flip-flops is entirely controlled from primary inputs. We consider the reset hardware to be a part of the combinational logic.

Lemma 1: A target single-fault that is untestable in $C(n)$ will also be untestable in $C(n+m)$, $n \geq 1$, $m \geq 1$.

Definition: A *target multifault* is a multiple fault in $C(n)$ such that the same single fault is introduced in each block.

Theorem 2 (Multifault Theorem): A target multifault that is untestable in $C(n)$ corresponds to an untestable single fault in the sequential circuit.

Proof: The circuit $C(n)$ with multifault can be viewed as a section of the iterative array model [20] of test generation. In $C(n)$, we assume completely controllable state inputs for the left block and completely observable state outputs for the right block. If a multifault is not detectable, then it implies that it is impossible to produce the fault effect either at any of the n primary outputs or at the state variable outputs of the right block. In other words, it is impossible to activate the fault by n vectors for any of the 2^q initial states, where q is the number of flip-flops in the circuit. Now if we place $C(n)$ in the iterative array model, we find that its state inputs may be only a subset of the possible 2^q states. Thus the faults that could not be activated in the isolated $C(n)$, still cannot be activated and, as a result, no test is possible. ■

Because of the similarity of $C(n)$ with multifault to the iterative array model of sequential circuit, the result of Theorem 2 may seem obvious to a reader. However, the main idea is that nondetectability of a multifault in a finite array implies nondetectability in *infinite array*. The notion of infinite array ($n \to \infty$) is used to represent the situation where state inputs of the left-most block are uncontrollable and state outputs of the right-most block are unobservable. Our earlier paper [3] on this work included a *lemma* stating that *a target multifault, untestable in* $C(n)$, *will also be untestable in* $C(n+m)$, $n \geq 1$, $m \geq 1$. As pointed out in a recent paper [23], that was not correct. Suppose, a multifault is untestable in $C(n)$. It is still possible that the next state output produced by $C(n)$ can allow fault activation in $C(n+m)$ resulting in the appearance of fault effect at some state variable. However, untestability in $C(n)$ does guarantee that the fault effect will not appear at a PO and will remain untestable in the sequential circuit according to Theorem 2.

Corollary: For $n = 1$, both theorems identify combinationally untestable (redundant) faults.

For infinite n, Theorem 2 can identify all untestable faults. Since the complexity of test generation grows exponentially with the circuit size, the case of small n is more practical. For small n, however, both theorems will only cover a partial set of untestable faults. There can be untestable faults identified by Theorem 1 that will not be identified by Theorem 2, and vice-versa. There can also be an overlap between the sets of untestable faults found according to the two theorems.

IV. Example

We will illustrate the use of Theorem 1 by an example. Fig. 3 shows a *modulo*-3 counter whose function is defined with three states which will call *valid states*. The valid states (Q_1Q_0) for this counter circuit are 00, 01, and 10. The reset input $R = 1$ sets the counter in 00 state. The output Z becomes 1 only when the counter is in 10 state. For the input $I = 0$, $R = 0$, the counter holds its state. When $I = 1$, $R = 0$, the state changes $00 \to 01 \to 10 \to 00 \cdots$ as flip-flops (FF) are clocked. The state 11, never reached in the normal operation of the counter, is defined as an *invalid state*. This state does exist in the circuit of Fig. 3, but is not reachable from any other state. The only way we can find the circuit in this state is through a possible power up.

All single stuck-at faults in the combinational logic (shown within the dotted-line box) are testable. The sequential circuit has six

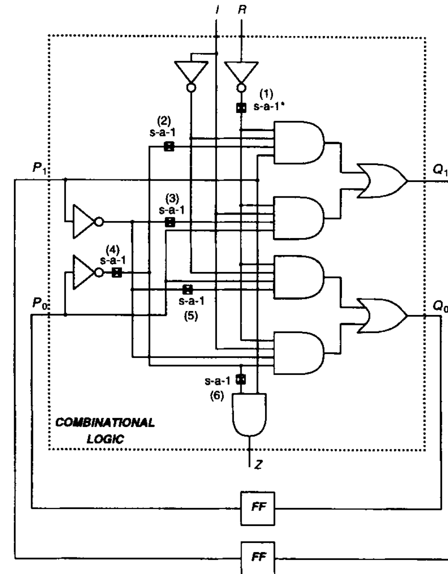

Fig. 3. Circuit of *modulo*-3 counter.

untestable faults. We used AT&T's GENTEST program [5] for combinational and sequential test generation. Sequentially untestable faults are marked in Fig. 3 as (1) through (6). Fault (1), shown with an asterisk, is potentially detectable. It prevents initialization of the circuit and is classified as potentially detectable by GENTEST. This is because the detection of the fault by the derived tests depends on the initial states of flip-flops when the circuit is *powered up*. However, the fault can be *definitely* tested by setting $I = 1$, $R = 1$, clocking the flip-flops, and observing the output at Z. The fault-free circuit will produce a steady 0000... at Z, but the faulty circuit will produce a periodic output ...0100100100... The starting bit of this sequence will depend upon the state in which the circuit powers up. Thus, the observation of a 1 in the output sequence implies the presence of a fault. However, the exact time when a 1 will be observed is nondeterministic. Finding such a test procedure is beyond the capability of GENTEST, which initializes both fault-free and faulty circuits, deterministically.

When we constructed $C(2)$ by duplicating the combinational logic of the counter circuit and generated tests for single stuck-at faults in the right block, faults (2), (3), (5), and (6) were found to be untestable by GENTEST. Increasing the size of the combinational array circuit ($n > 2$) did not identify any more untestable faults. Thus, Theorem 1 correctly finds four out of five untestable faults in this example.

V. Application

An untestability checker based on Theorem 1 is easy to implement since it requires a single-fault combinational ATPG program. All results given in this paper only deal with this method. Several ISCAS benchmarks were analyzed.

We wrote an *awk* program [4] to generate the networks $C(n)$. Table I shows the test generation results for some of the ISCAS '89 benchmarks. The array size, n, was progressively increased. The faults identified as untestable were dropped from consideration before going to the next higher value of n. The process was stopped if 2 or 3 consecutive increases in n did not produce any new untestable fault. For each n, Table I gives the CPU time of GENTEST on SUN Sparc 2. The untestable faults shown are the cumulative numbers. Notice

TABLE I
IDENTIFICATION OF UNTESTABLE FAULTS BY THEOREM 1 USING GENTEST [5]

Circuit	$n = 1$		$n = 2$		$n = 3$		$n = 4$		$n = 5$		$n = 6$		$n = 7$		$n = 8$	
Name	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts
s208	0.8	0	1.3	19	2.1	50	2.7	50	3.3	50	-	-	-	-	5.3	50
s298	0.9	0	1.3	0	3.1	11	5.2	11	13.6	20	74.4	21	450.4	26	1473.4	26
s344	1.2	0	2.1	1	3.4	5	4.0	5	-	-	-	-	-	-	-	-
s382	1.4	0	4.2	4	3.5	4	4.6	4	7.4	4	-	-	-	-	-	-
s386	2.0	0	10.2	70	39.3	70	-	-	-	-	-	-	-	-	-	-
s820	8.6	0	21.1	27	-	-	333.3	27	-	-	-	-	-	-	-	-
s5378	98.8	40	174.1	335	589.1	470	385.8	564	680.8	671	851.9	774	649.3	781	-	-

TABLE II
INFLUENCE OF UNTESTABLE FAULT IDENTIFICATION ON TEST GENERATION

Circuit Name	Total No. of Faults	Without untestability identification			With untestability identification		
		No. of Vectors	Coverage Efficiency (%)	CPU s	No. of Vectors	Coverage Efficiency (%)	CPU s
s208	257	158	98.1	57.3	160	99.2	47.2
s298	312	218	91.0	792.8	248	97.1	528.6
s344	360	159	97.2	276.7	138	97.8	105.2
s382	413	706	87.2	1247.0	706	88.1	1175.0
s386	386	211	100.0	2022.0	201	100.0	363.3
s820	850	413	84.0	27005.4	391	88.8	24235.9
s5378	4603	436	76.2	89247.0	384	85.5	5757.5

that many of the smaller circuits have no combinational ($n = 1$) redundancy. That is generally not true for large circuits.

Since Theorem 1 is based only on a *necessary* condition for testing, it does not identify all untestable faults. Except for circuits s298 and s5378, we found that the number of identified faults did not increase for $n \geq 3$. For s386, all 70 untestable faults were found for $n \geq 2$. Since the test generator was used in the combinational mode, the CPU time is reasonably small. This can be further reduced by using improved combinational circuit test generators [6].

Table II shows the results of sequential test generation by GENTEST [5]. The first set of results was obtained by attempting test generation for all faults. In the second set, the largest number of untestable faults (shown in Table I) were removed from the fault list prior to test generation. The test generator identified a few additional untestable faults. The *coverage efficiency* is computed by including the identified untestable faults among detected faults. In every case, there is an improvement in the coverage efficiency. The reported CPU time is for a SUN Sparc 2. Improvement in sequential test generation efficiency by identification of untestable faults has also been reported by other authors [15].

Examine the case of s5378. In Table I, 781 untestable faults were found after seven runs of the combinational ATPG. The total CPU time is 98.8 + 174.1 + 589.1 + ... = 3439.8 s. When these faults were removed from the fault list, test generation for the sequential circuit required 5757.5 s to generate 384 vectors. Additional 11 untestable faults were identified by the sequential ATPG bringing the number of known untestable faults to 792. Counting these among the detected faults, the coverage was 85.5%. In comparison, when the 781 untestable faults were not removed, the sequential ATPG used 89247 s of CPU time to cover 76.2% faults. In this process, it identified only 359 untestable faults.

Similar improvements in the efficiency of sequential circuit ATPG have been reported by other workers. Their methods of identifying untestable faults differ in many respects. Liang *et al.*, use symbolic simulation to determine uninitializable states [15]. Gouders and Kaibel [11] use symbolic state justification to find untestable faults in circuits with feedbacks. They also use a learning mechanism where unjustifiable states are stored and reused. Chen and Bushnell [9] devise a more elaborate learning procedure in which an unjustifiable state can be decomposed into several new unjustifiable states. The reader is advised to compare these techniques since such details are beyond the scope of this paper. A common feature of the techniques we just cited is that they identify untestable faults in the framework of sequential ATPG. The main idea that makes the technique of this paper different is that we transform the sequentially untestable faults to a combinational model.

In several cases, we noticed that the sequential ATPG found untestable faults that were not discovered by the single-fault combinational ATPG Theorem. Obviously, the theorem does not guarantee to find all untestable faults. GENTEST uses *backward* time processing [5]. It, therefore, establishes fault propagation before attempting state justification. Notice that fault propagation is addressed by the multifault ATPG method (Theorem 2), whose examples may be found in a recent paper [23]. Even so, because of finite n in practice, there may always be some untestable faults that are not identified by any of the theorems, but are found by a sequential ATPG.

TABLE III
IDENTIFICATION OF UNTESTABLE FAULTS BY THEOREM 1 USING TRAN [6]

Circuit Name	Total No. of Faults	$n = 1$		$n = 2$		$n = 3$		$n = 4$		$n = 5$		$n = 6$		$n = 7$		$n = 8$	
		CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts	CPU s	Unt. Flts
s349	350	0.2	2	0.4	3	1.0	7	1.4	7	1.5	7	2.3	7	4.9	9	2.7	9
s400	424	0.2	6	0.5	10	1.0	10	2.9	10	8.1	10	14.1	10	22.0	10	27.8	10
s420	430	0.5	0	1.1	61	2.6	184	2.5	221	1.6	221	2.1	221	2.4	221	2.5	221
s444	474	0.2	14	0.6	18	1.2	18	3.1	18	10.5	18	11.6	18	26.8	18	38.6	18
s510	564	0.1	0	0.4	0	0.1	0	0.2	0	0.3	0	0.4	0	2.8	0	0.5	0
s526	555	0.4	1	1.5	7	4.7	16	24.6	16	79.0	25	151.5	26	223.1	27	284.8	27
s526n	553	0.3	0	1.5	6	4.7	15	24.6	15	86.0	24	93.3	25	167.8	26	231.0	26
s641	467	0.8	0	2.5	0	4.7	0	8.4	0	13.2	0	19.9	0	30.4	0	43.3	0
s713	581	1.5	38	2.7	38	4.7	38	9.7	38	15.0	38	21.7	38	29.6	38	43.3	38
s832	870	0.9	14	19.7	51	49.9	51	99.2	51	192.3	51	349.8	51	563.4	51	635.0	51
s838	857	3.4	0	6.0	141	16.0	432	39.1	549	4.3	549	5.2	549	6.2	549	7.2	549
s953	1079	0.9	0	11.0	10	4.9	10	9.4	10	15.7	10	30.8	10	38.4	10	98.6	10
s1196	1242	2.0	0	3.6	0	5.2	0	6.6	0	8.7	0	8.8	0	12.1	0	10.3	0
s1238	1355	4.6	69	4.3	69	5.9	69	7.9	69	9.8	69	9.6	69	11.7	69	12.6	69
s1423	1515	2.2	14	5.8	14	17.4	14	50.0	14	98.6	14	185.9	14	263.9	14	376.0	14
s1488	1486	1.1	0	30.3	40	98.8	40	390.7	40	1493.9	40	3496.1	40	6182.6	40	10088.4	40
s1494	1506	1.2	12	28.4	51	102.0	51	318.6	51	1281.8	51	3882.7	51	5994.5	51	9302.1	51
s9234	6927	169.5	452	845.7	520	1778.4	524	-	-	-	-	-	-	-	-	-	-
s13207	9815	205.6	151	885.9	924	3237.2	961	-	-	-	-	-	-	-	-	-	-
s15850	11725	456.6	389	900.8	440	2840.1	448	-	-	-	-	-	-	-	-	-	-
s35932	39094	267.7	3984	80.8	3984	137.9	3984	-	-	-	-	-	-	-	-	-	-
s38417	31180	754.9	165	1554.9	203	4685.7	391	-	-	-	-	-	-	-	-	-	-
s38584	36303	452.1	1506	3573.4	2131	5050.8	2142	-	-	-	-	-	-	-	-	-	-

Table III gives additional results for ISCAS'89 benchmarks obtained by using the combinational ATPG program, TRAN [6]. All CPU times are for SUN Sparc 2. Seventeen smaller circuits were expanded up to $n = 8$. These are separated by a line next to s1494 in the table. For three circuits (s510, s641, and s1196) no untestable fault was found by this method. In three other circuits (s713, s1238, and s1423), only combinationally untestable faults were found. In six, no untestable faults were located beyond $n = 2$. Remaining five circuits had untestable faults identified for $n \leq 7$. Due to long run times, the six largest circuits were only run with $n \leq 3$.

VI. CONCLUSION

We believe that there is need for simple algorithms for identifying untestable faults in sequential circuits. Since the problem is complex, approximate methods are desirable. The theorems presented in this paper allow the use of combinational test generator for identifying untestable faults in sequential circuits. In general, many of the untestable faults may be sequentially redundant. Further analysis is required to identify them. The usefulness of the theorems stems from the fact that a combinational ATPG program is much simpler than a sequential ATPG program. Our experience shows that using a reasonably small arrays of eight or fewer blocks we can find many untestable faults with Theorem 1. The sequential ATPG program simply abandons on many of the same faults in spite of fairly large time (or backtrack) limit.

To use the proposed theorems, the availability of an ATPG program is not necessary. Any procedure that finds redundancies in combinational circuits can be extended to find untestable faults in sequential circuits [13], [14], [18]. Application of the theorems is also possible in conjunction with state-enumeration procedures [10], [16], where only the legal states may be allowed to appear at the state inputs of the left-most block. However, the *reset state*, if present, should be explicitly included in the state transition graph. A recent paper [15] presents quite a different method for identifying untestable faults in sequential circuits. The authors use symbolic simulation to determine some of the *impossible* states. Their method does not use ATPG. However, it cannot identify combinationally untestable faults. It will be interesting to compare the set of untestable faults found by the symbolic simulation method with that found by techniques presented in this paper.

REFERENCES

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design.* New York: Comput. Sci. Press, W. H. Freeman, 1990.

[2] M. Abramovici and P. S. Parikh, "Warning: 100% fault coverage may be misleading!!," in *Proc. Int. Test Conf.,* Sept. 1992, pp. 662–668.

[3] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG theorems for identifying untestable faults in sequential circuits," *Proc. Euro. Test Conf.,* pp. 249–253, Apr. 1993.

[4] A. V. Aho, B. W. Kernighan, and P. J. Weinberger, *The AWK Programming Language.* Reading, MA: Addison-Wesley, 1988.
[5] B. Bencivenga, T. J. Chakraborty, and S. Davidson, "GENTEST: The architecture of sequential circuit test generator," in *Proc. Custom Integrat. Circuits Conf.,* May 1991, pp. 17.3.1–17.3.4.
[6] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A transitive closure algorithm for test generation," *IEEE Trans. Computer-Aided Design,* vol. 12, pp. 1015–1028, July 1993.
[7] S. T. Chakradhar, S. G. Rothweiler, and V. D. Agrawal, "Redundancy removal and test generation for circuits with non-Boolean primitives," in *Proc. 13th IEEE VLSI Test Symp.,* Apr.–May 1995, pp. 12–19.
[8] K.-T. Cheng, "On removing redundancy in sequential circuits," in *Proc. 28th Design Automat. Conf.,* June 1991, pp. 164–169.
[9] X. Chen and M. L. Bushnell, "Dynamic state and objective learning for sequential circuit automatic test generation using decomposition equivalence," in *Proc. 24th Fault Tolerant Computing Symp.,* 1994, pp. 446–455.
[10] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration," *IEEE Trans. Computer-Aided Design,* vol. 12, pp. 935–945, July 1993.
[11] N. Gouders and R. Kaibel, "Test generation techniques for sequential circuits," in *Proc. 9th IEEE VLSI Test Symp.,* 1991, pp. 221–226.
[12] M. A. Iyer and M. Abramovici, "Identifying sequential redundancies without search," Tech. Memo., AT&T Bell Labs., Murray Hill, NJ, June 17, 1994.
[13] ——, "Low-cost redundancy identification for combinational circuits," in *Proc. 7th Int. Conf. VLSI Design,* Jan. 1994, pp. 315–318.
[14] ——, "Sequentially untestable faults identified without search," in *Proc. Int. Test Conf.,* 1994, pp. 259–266.
[15] H.-C. Liang, C. L. Lee, and J. E. Chen, "A sequential redundant fault identification scheme and its application to test generation," in *Proc. Third Asian Test Symp.,* 1994, pp. 57–62.
[16] D. E. Long, M. A. Iyer, and M. Abramovici, "Identifying sequentially untestable faults using illegal states," in *Proc. 13th IEEE VLSI Test Symp.,* Apr.–May 1995, pp. 4–11, .
[17] H.-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test generation for sequential circuits," *IEEE Trans. Computer-Aided Design,* vol. 7, pp. 1081–1093, Oct. 1988.
[18] P. R. Menon, H. Ahuja, and M. Harihara, "Redundancy identification and removal in combinational circuits," *IEEE Trans. Computer-Aided Design,* vol. 13, pp. 646–651, May 1994.
[19] A. Miczo, *Digital Logic Testing and Simulation.* New York: Harper & Row, 1986.
[20] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. Comput.,* vol. C-25, pp. 630–636, June 1976.
[21] I. Pomeranz and S. M. Reddy, "Test generation for synchronous sequential circuits using multiple observation times," in *Proc. 21st Fault Tolerant Computing Symp.,* 1991, pp. 52–59.
[22] ——, "Classification of faults in synchronous sequential circuits," *IEEE Trans. Comput.,* vol. 42, pp. 1066–1077, Sept. 1993.
[23] ——, "On identifying undetectable and redundant faults in synchronous sequential circuits," in *Proc. 12th IEEE VLSI Test Symp.,* Apr. 1994, pp. 8–14.

Pseudo-Exhaustive Built-In TPG for Sequential Circuits

Dimitrios Kagaris, Spyros Tragoudas, and Dinesh Bhatia

***Abstract*— We address the issue of pseudo-exhaustive test pattern generation (TPG) for the built-in self-test (BIST) of sequential circuits. Let d be the sequential depth, and w be the input dependency limit. We use an LFSR/SR Test Pattern Generator and a small additional hardware overhead to automatically generate $d \cdot 2^w$ test patterns to test the circuit pseudo-exhaustively or, alternatively, pseudo-randomly with less hardware overhead and extremely high fault coverage. Our scheme uses novel retiming algorithms and transforms the circuit to an equivalent (for test purposes) one by scanning a subset of flip-flops for breaking its cyclic structure, bounding the sequential depth, forcing the input dependency limit, balancing the circuit, and maintaining the clock period. We present the first polynomial time algorithm to bound the sequential depth of a circuit by retiming with minimum number of flip-flops and subject to a clock period bound. We also give a retiming-based polynomial time algorithm to balance a circuit by inserting a minimum number of bypass delay cells. Experimental results on the ISCAS'89 benchmarks indicate that our method outperforms a previously proposed approach, which not only does not provide for on-chip test pattern generation but also requires $O(q \cdot f \cdot 2^w)$ test patterns, where q is the total number of primary or pseudo-primary outputs in the circuit and f is the total number of flip-flops.**

I. Introduction

Testing sequential circuits presents many difficulties. A standard first step to alleviate the testability problem in sequential circuits is to incorporate some of their flip-flops into a *scan chain* [1]. Due to the hardware cost of the conversion of a flip-flop into a scan element, the number of selected flip-flops must be small. For this reason, the scan chain is specified as *partial*, in contrast to a *full* scan chain that includes every flip-flop in the circuit.

We refer to the set of the selected flip-flops as PS (Partial Scan) set. The efficacy of a particular PS set is judged by how much it facilitates test pattern generation (TPG) and how high a fault coverage it yields. The topology of the flip-flops of the circuit is given by its S-graph [3], [14]–[16], [20]. The nodes of the S-graph are the flip-flops of the circuit, and two nodes u, v are joined by a directed edge (u, v), iff there is a path in the circuit graph from flip-flop u to flip-flop v containing no other flip-flop (see Fig. 1). A good criterion for obtaining an effective PS set is to select a minimum set FV of flip-flops that make the S-graph acyclic [3], [16]. Since the corresponding graph-theoretic problem (Minimum Feedback Vertex Set [7]) is NP-complete, a number of heuristic solutions have been proposed [3], [5], [14]–[16]. In addition, the efficacy of a PS set is enhanced, if the PS set also contains appropriate flip-flops, so that the sequential depth, defined as the longest path in the acyclic S-graph, is small [3], [16].

Even if the S-graph is acyclic, test pattern generation for the resulting circuit is still more complicated than the testing of a purely combinational circuit. The problem is that each time a new test pattern

Manuscript received September 20, 1993; revised March 23, 1994 and October 26, 1994. This paper was recommended by Associate Editor T. Cheng. This work was supported by Grants from the National Science Foundation (NSF MIP-9409905) and a 1993–1994 Design Automation Scholarship. This paper was presented in part at the IEEE International Conference on Computer Design, 1993.

D. Kagaris is with the Department of Electrical Engineering, Southern Illinois University, Carbondale, IL 62901 USA.

S. Tragoudas is with the Department of Computer Science, Southern Illinois University, Carbondale, IL 62901 USA.

D. Bhatia is with the Department of Electrical and Computer Engineering, University of Cincinnati, Cincinnati, OH 45221 USA.

IEEE Log Number 9412315.

0278–0070/95$04.00 © 1995 IEEE