Tutorial and Survey Paper:

Gate-Level Test Generation for Sequential Circuits

KWANG-TING CHENG
University of California, Santa Barbara

This paper discusses the gate-level automatic test pattern generation (ATPG) methods and techniques for sequential circuits. The basic concepts, examples, advantages, and limitations of representative methods are reviewed in detail. The relationship between gate-level sequential circuit ATPG and the partial scan design is also discussed.

Categories and Subject Descriptors: B.7.3 [**Integrated Circuits**]: Reliability and Testing—*test generation*

General Terms: Algorithms, Reliability, Verification

Additional Key Words and Phrases: Automatic test generation, IC testing, sequential circuit test generation

1. INTRODUCTION

The first automatic test pattern generation (ATPG) algorithm for sequential circuits was reported by Seshu and Freeman [1962]. Since then, tremendous progress in the development of algorithms and tools has been made. One of the earliest commercial tools, LASAR [Thomas 1971], was reported in the early seventies, and currently there are more than a dozen sequential circuit test generators commercially available. Due to the high complexity of the sequential circuit test generation problem, it remains a challenging task for these tools to automatically generate high quality tests for large, highly sequential circuits that do not incorporate any design for testability (DfT) scheme. However, these test generators combined with low-overhead DfT techniques such as partial scan have shown a certain success in testing large designs. For designs that are sensitive to area and/or performance overhead, the solution of using sequential circuit ATPG

Author's address: Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106. ⟨timcheng@ece.ucsb.edu⟩
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 1996 ACM 1084-4309/96/1000–0405 $03.50

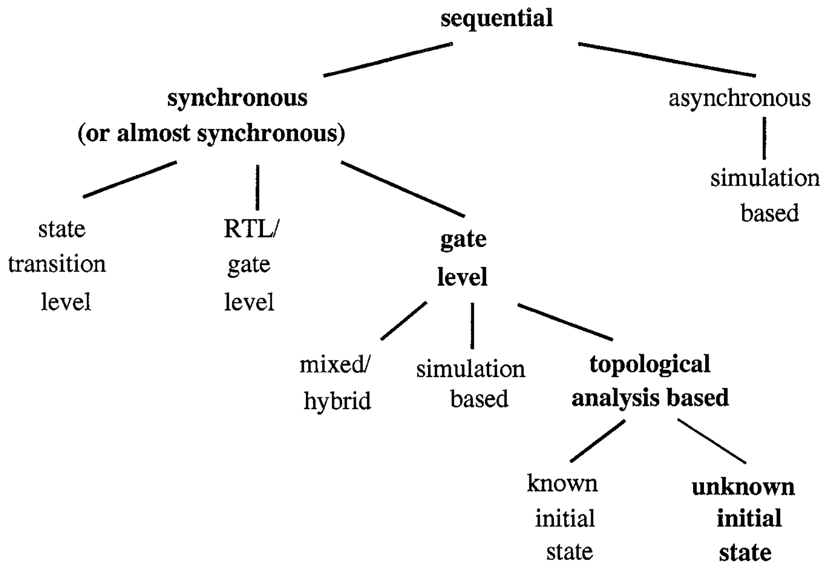

Fig. 1. Sequential test generation: Taxonomy.

and partial scan offers an attractive alternative to the popular full-scan solution, which is based on combinational circuit ATPG.

It requires a sequence of vectors to detect a single stuck-at fault in a sequential circuit. Also, due to the presence of the memory elements, the controllability and observability of the internal signals in a sequential circuit are, in general, much worse than those in a combinational circuit. These factors cause the complexity of sequential circuit test generation to be much higher than that for combinational circuits. The test generators for sequential circuits search for a sequence of vectors to detect a particular fault through the space of all possible vectors. Various search strategies and heuristics have been devised to find a shorter sequence and/or to find a faster sequence. However, according to reported results, no single strategy/heuristic outperforms others for all applications/circuits. This observation implies that a test generator should include a comprehensive set of test generation procedures. In this paper, we will discuss the basics and give a survey of methods and techniques for sequential circuit test generation for single stuck-at faults. We focus on the methods that are based on gate-level circuit models. Examples will be given to illustrate the basics of representative methods. We assume that readers are familiar with the basics of combinational circuit test generation, which are not reviewed in this paper. More general tutorials/surveys on test generation (including techniques for combinational circuits and for other fault models) can be found in Agrawal and Seth [1988] and Abraham and Agrawal [1986].

Figure 1 shows the taxonomy for sequential test generation approaches. Few approaches can directly deal with the timing issues present in highly asynchronous circuits. Most sequential circuit test generation approaches

neglect circuit delays during test generation. Such approaches primarily target synchronous or *almost synchronous* (i.e., with some asynchronous reset/clear and/or few asynchronous loops) sequential circuits and cannot properly handle highly asynchronous circuits whose functions are strongly related to the circuit delays and are sensitive to races and hazards. One engineering solution to using such approaches for asynchronous circuits is to divide the test generation process into two phases. A potential test is first generated by ignoring the circuit delays. The potential test is then simulated using proper delay models in the second phase to check its validity. If the potential test is invalid due to races, hazards, or oscillations, test generation is called again for a new potential test. The approaches for (almost) synchronous circuits can be classified according to the level of abstraction at which the circuit is described. One class of approaches uses the state transition graph for test generation [Hennie 1964; Hsieh 1971; Cheng and Jou 1992; Pomeranz and Reddy 1991]. This class is suitable for pure controllers for which the state transition graphs are either readily available or easily extractable from a lower-level description. For data-dominated circuits, if both register transfer level (RTL) and gate-level descriptions are provided, several approaches can effectively use the RTL description for state justification and fault propagation [Hill and Huey 1977; Breuer and Friedman 1980; Ghosh 1991]. Most of the commercial test generators are based on the gate-level description. They either employ the iterative array model [Kubo 1968; Putzolu and Roth 1971] and use topological analysis algorithms [Martlett 1978; Cheng 1989; Niermann and Patel 1991; Kelsey et al. 1993], or are enhanced from a fault simulator [Snethen 1977; Cheng 1989; Saab et al. 1992; Rudnick et al. 1994; Prinetto et al. 1994] or use the mixed/hybrid methods that combine the topological-analysis-based methods and the simulation-based methods [Saab et al. 1994; Rudnick and Patel 1995; Hsiao et al. 1996]. Most gate-level approaches assume an unknown initial state in the flip-flops, although some approaches assume a known initial state to avoid initialization of the memory elements [Ma et al. 1988; Ghosh et al. 1991; Cho et al. 1993]. The highlighted models and approaches in Figure 1 are those commonly adopted in most of today's vendor tools.

The paper is organized as follows. In Section 2, we describe the basic principles of the iterative array model and an in-depth survey of the topological-analysis-based algorithms. We then discuss the simulation-based methods and the hybrid methods in Section 3. A special class of approaches that assumes a known initial state in the flip-flops is discussed in Section 4. In Section 5, the relationship of sequential circuit test generation and partial scan is discussed. A brief introduction to the approaches based on circuit models at higher levels of abstraction is given in Section 6. We conclude by introducing the concept of functional learning and pointing out the potential of using it to improve the performance and robustness of sequential circuit ATPG.

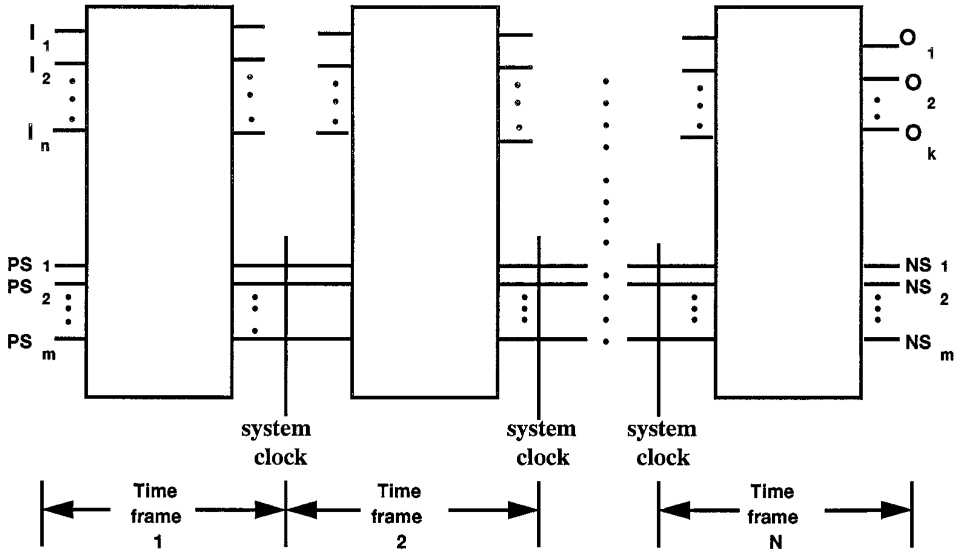

Fig. 2. The iterative-array model of a sequential circuit.

2. TOPOLOGICAL-ANALYSIS-BASED APPROACHES

The iterative array model. Many sequential circuit test generators have been devised on the basis of fundamental combinational algorithms. As shown in Figure 2, a combinational model for a sequential circuit is constructed by regenerating the feedback signals from previous-time copies of the circuit. A rectangle in Figure 2, called time-frame, represents a copy of the combinational portion of the circuit. The inputs of a time-frame include the primary inputs (I_i, $i = 1, \ldots, n$), and the outputs of the flip-flops, called the present state lines (PS_i, $i = 1, \ldots, m$). The outputs of a time-frame include the primary outputs (O_i, $i = 1, \ldots, k$), and the data inputs of the flip-flops, called the next state lines (NS_i, $i = 1, \ldots,$ m). The clock signal is not included in the primary input list in this model. A clock pulse needs to be applied between two time-frames to update the values at the present state lines of one time-frame from the next state lines of its previous time-frame. This combinational model is used to approximate the timing behavior of the circuit. Topological analysis algorithms that activate faults and sensitize paths through these multiple copies of the combinational circuit are then used to generate tests. Note that a single stuck-at fault in a sequential circuit will correspond to a multiple stuck-at fault in the iterative array model where each time-frame contains a stuck-at fault at the fault site.

Extended D Algorithm

The earliest algorithms extended the D-algorithm [Roth 1966] for the iterative array model [Kubo 1968; Putzolu and Roth 1971]. It starts with one copy of the combinational logic and sets it to time-frame 0. The

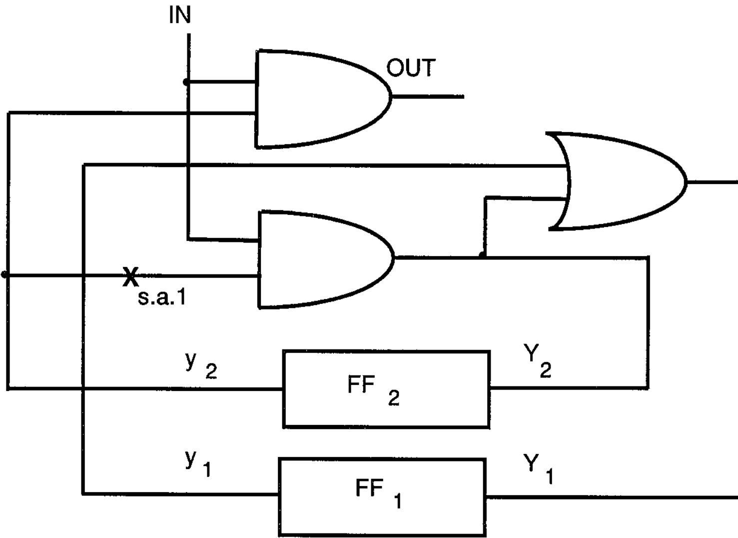

Fig. 3. An example.

D-algorithm is used for time-frame 0 to generate a combinational test. When the fault effect is propagated to the next state lines, a new copy of the combinational logic is created as the next time-frame and the fault propagation continues. When there are values required at the present state lines, a new copy of the combinational logic is created as the previous time-frame. The state justification is then performed backwards in the previous time-frame. The process continues until there is no value requirement at the present state lines and a fault effect appears at a primary output.

Example 1. Consider the synchronous sequential circuit in Figure 3. A copy of the combinational portion of the circuit, called time-frame 0, is shown in Figure 4. After the D-algorithm is applied for this time-frame, a fault effect $\bar{D}$ (i.e., 0/1—the value of the fault-free circuit is 0 and that of the faulty circuit is 1) appears at the next state line Y_2 and a value 0 is required at the present state line y_2. A new copy of the combination logic, called time-frame 1, is attached to the right of time-frame 0 (fig. 5) for further fault propagation. The fault effect can then be successfully propagated to the primary output OUT in time-frame 1. The value assigned at y_2 in time-frame 0 needs justification. Another copy of the combinational logic, called time-frame -1, is attached to the left of time-frame 0 (fig. 6) and the value at y_2 of time-frame 0 can be successfully justified by assigning a 0 at the primary input in time-frame -1. Therefore, the test sequence is successfully generated. Regardless of the initial states of the sequential circuit, sequence {IN = 0, clock, IN = 1, clock, IN = 1} can always produce a fault effect at the primary output OUT for the target fault.

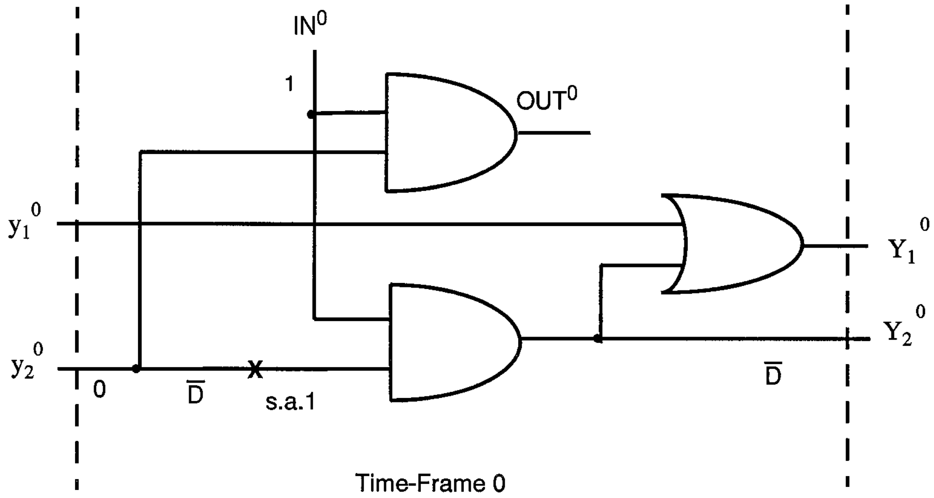

Fig. 4. Step 1.

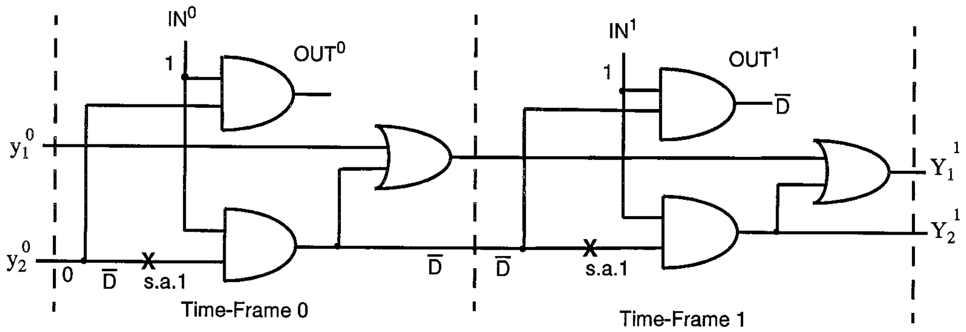

Fig. 5. Step 2.

The Nine-Valued Logic

Muth [1976] pointed out that the five-valued logic used in the D-algorithm is not appropriate for sequential circuits. A nine-valued logic is suggested to take into account the possible repeated effects of the fault in the iterative array model. Each of the nine values is defined by an ordered pair of binary values—the first value of the pair represents the binary value of the fault-free circuit and the second value represents the binary value of the faulty circuit. The binary value of a circuit could be 1, 0, or X (don't care) and therefore there are nine distinct ordered pairs (0/0, 0/1, 0/X, 1/0, 1/1, 1/X, X/0, X/1, and X/X).

Example 2. Consider the circuit in Figure 7. The signal values assigned using the extended D-algorithm are shown in Figure 8. The process starts with time-frame 1 and requires a 0 at PS y. Further justification in time-frame 0 eventually leads to a conflict at primary input a on which a 0 is required while the signal is stuck-at-1. It thus mistakenly concludes that

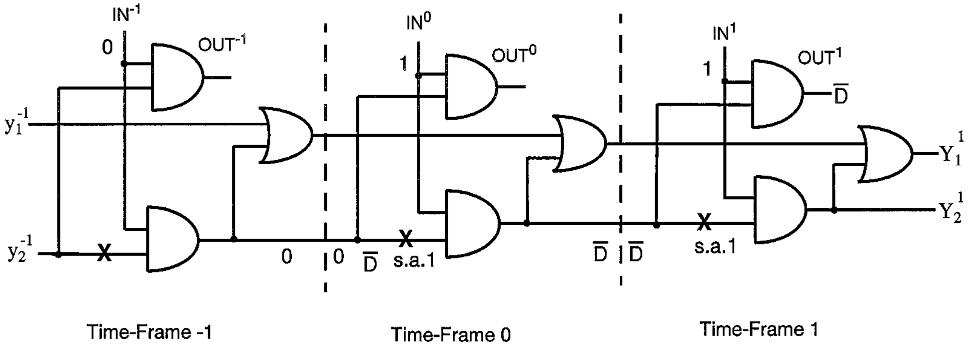

Fig. 6. Step 3.

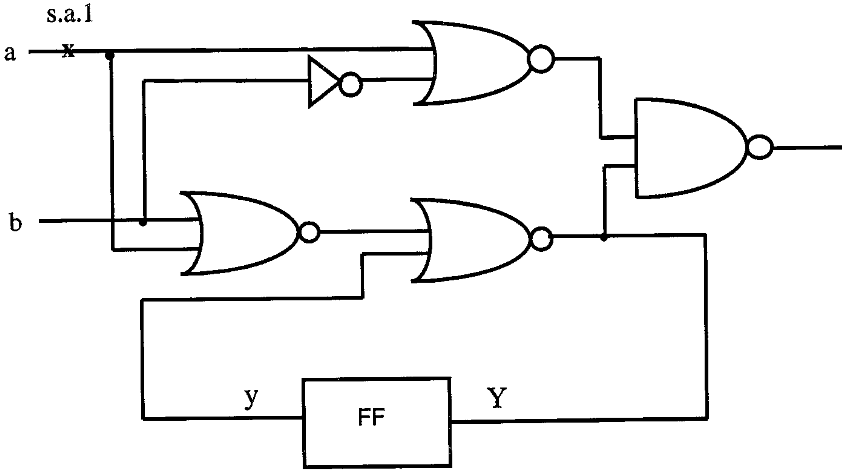

Fig. 7. An example for 9-valued test generation.

there is no 2-vector test sequence for the target fault. Figure 9 shows the values assigned in the test generation process using the nine-valued logic. To propagate the fault effect from primary input a to the output of gate g_1 in time-frame 1, the only requirement is that the other input of g_1 have a 0 *for the fault-free circuit* (there is no value requirement at this signal for the faulty circuit). Therefore, under the nine-valued model, the required value is 0/X. Eventually, this leads to a value requirement 0/X at present state line y. Due to this relaxed requirement, no conflict occurs at a in time-frame 0 and a 2-vector test $\{ab = 00, 01\}$ for the target fault is successfully found. (Note that a clock pulse needs to be applied between these two vectors. In the rest of the paper, the clock will not be explicitly shown in any test sequence and it is assumed that a clock pulse will be inserted between any two vectors.) The reason why the five-valued model fails to find the test is because it *over-specifies* the value requirements at some internal nodes. For this example, the required value at present state line y

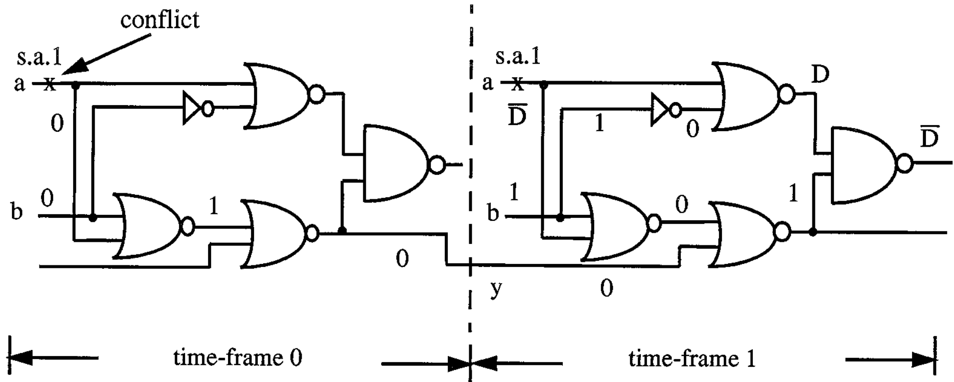

Fig. 8. Test generation using 5-valued logic.

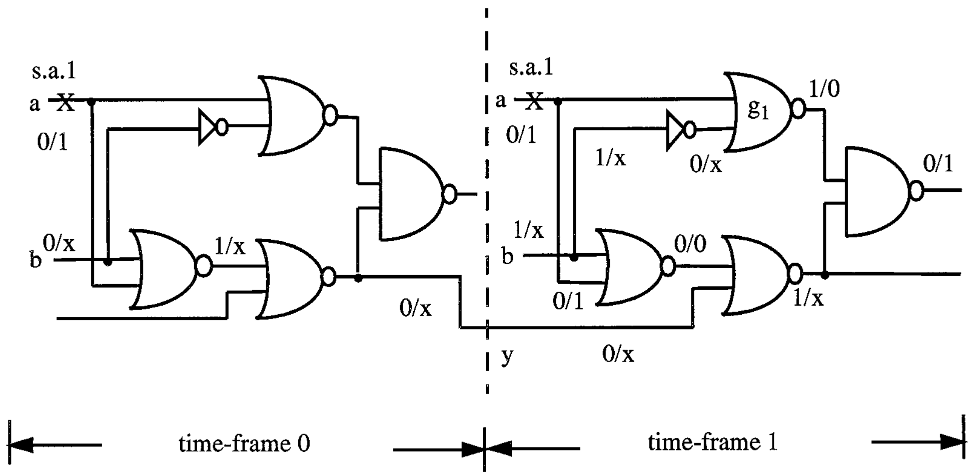

Fig. 9. Test generation using 9-valued logic.

in time-frame 1 is over-specified under the five-valued model, that is, a 0 at y for the faulty circuit is unnecessary. It happens in this example that the over-specified requirement cannot be justified without a requirement at y in time-frame 0. On the other hand, the nine-valued model precisely represents the requirement that can be successfully justified at time-frame 0 without further requirement at y.

Reverse Time Processing

The extended D-algorithm and the nine-valued algorithm use *mixed forward and reverse time processing* techniques during test generation. The requirements created during the forward process (fault propagation) have to be justified by the backward process later. The mixed time processing techniques have some disadvantages: (1) The test generator may need to maintain a large number of time-frames during test generation because all time-frames are partially processed and (2) the implementation is some-

what complicated. The Reverse Time Processing (RTP) technique used in the Extended Backtrace Algorithm (EBT) [Martlett 1978] overcomes the problems caused by the mixed time processing technique. RTP works backwards in time from the last time-frame to the first time-frame. For a given fault, it pre-selects a path from the fault site to a primary output. This path may involve several time-frames. The selected path is then sensitized backwards starting from the primary output. If the path is successfully sensitized, backward justification is continued for the required value at the fault site. If the sensitization process fails, another path is selected. The advantages of RTP include: (1) At any time during the test generation process, only two time-frames need to be maintaine: the current time-frame and the previous time-frame. For such a uni-directional algorithm, the backward justification process is done in a breadth-first manner. The value requirements in time-frame n are completely justified before starting the justification of the requirements in the previous time-frame $n - 1$. Therefore, the justified values at internal nodes of time-frame n can be discarded when the justification of time-frame $n - 1$ starts. As a result, the memory usage is low and the implementation is easier. Note that the decision points and their corresponding circuit status still need to be stacked for the purpose of backtracking. (2) It is easier to identify repetition of state requirements. A state requirement is defined as the state specified at the present state lines of a time-frame during the backward justification process. If a state requirement has been visited earlier during the current backward justification process, the test generator has found a loop in the state transition diagram. We call such a situation *state repetition*. The backward justification process should not continue to circle that loop and backtracking should take place immediately. In reverse time processing, justification in time-frame n is completed before the beginning of justification in time-frame $n - 1$. Therefore, simply by recording the state requirement after the completion of backward justification of each time-frame and comparing each newly visited state requirement with the list of previously visited state requirements, state repetition can be easily identified and, thus, the search can be conducted more effectively. Similarly, the test generator can maintain a list of illegal states, the states that have been previously determined as unjustifiable. Each newly visited state requirement should also be compared against this list to determine whether the state requirement is an identified illegal state to save repetitive and unnecessary searches.

There are two major problems with the EBT algorithm: (1) Only a single-path is selected for sensitization. Faults that require multiple-path sensitization for detection may not be covered. (2) The number of possible paths from the fault site to the primary outputs is very large; trying path by path is not practical.

The BACK Algorithm

The BACK algorithm [Cheng 1988] is an improvement of the EBT algorithm. It also employs the RTP technique. Instead of pre-selecting a path,

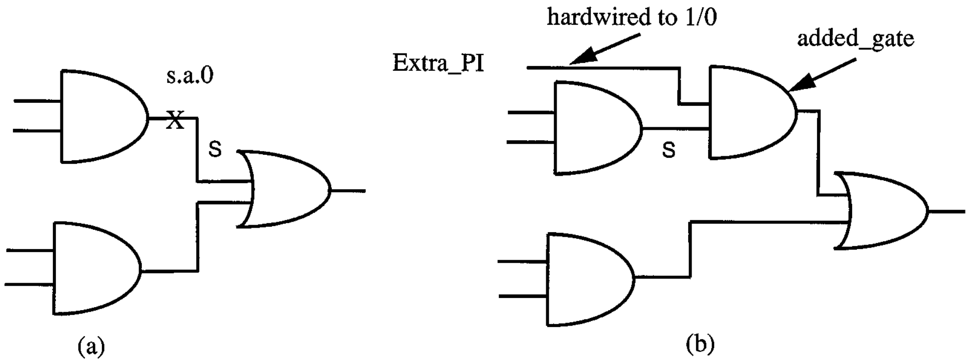

Fig. 10. Fault injection using an extra gate. (a) Before fault injection; (b) after fault injection.

the BACK algorithm pre-selects a primary output. It assigns a D or $\bar{D}$ to the selected primary output and justifies the value backwards. A testability measure (called *drivability*) is used to guide the backward D-justification from the selected primary output to the fault site. Drivability is a measure associated with a signal that estimates the effort of propagating a D or $\bar{D}$ from the fault site to the signal. The drivability measurement is derived based on the SCOAP [Goldstein 1979] controllability measurement of both fault-free and faulty circuits. For a given fault, the drivability measure of each signal is computed before test generation starts. In the next two paragraphs, the definition and the calculation procedure of drivability measures will be summarized.

Drivability. To describe the details of drivability calculation, a fault injection technique [Niermann et al. 1992] used in the BACK algorithm needs to be explained first. Traditionally, fault injection is accomplished by associating a bit flag with each signal indicating whether the signal is a stuck-at faulty signal. This method requires that the flag be examined for every signal justification and signal implication, even though only one signal (the target fault) is faulty in an ATPG run. Instead of using such a software method for fault injection, an extra 2-input gate is inserted into the circuit for injecting a fault. To inject a stuck-at-0 fault at signal s, an extra 2-input AND gate is inserted as shown in Figure 10. One of the inputs of the added gate is an extra primary input whose value is hardwired to 1/0 to reflect the correct functions of the fault-free and faulty circuits. Similarly, to inject a stuck-at-1 fault, an extra OR gate is inserted and the value of the extra primary input is hardwired to 0/1. The advantages of this fault-injection technique are that all gates can be treated equally and it eliminates the need for flag checking.

The controllabilities of the fault-free and faulty circuits are different and are computed separately. For the fault-free circuit, the SCOAP 1-controllability CC_g^1 (s) and 0-controllability CC_g^0 (s) of each signal s can be computed in the preprocessing stage using the procedure described in Goldstein [1979]. For the faulty circuit of a target fault, the signal controllabilities

```
1. Calculate controllability measures for the fault-free circuit.
2. Select a target fault and calculate controllability measures
   for the faulty circuit.
3. Derive drivability measures.
4. Select a primary output with the smallest drivability measure
   and assign a D or D̄. If all primary outputs with finite
   drivability have been considered, report the fault as
   undetectable and go to Step 2.
5. Justify all values required at the current time-frame:
   a) If a decision is made, store the decision in the decision
      stack and store the current circuit status.
   b) If a conflict occurs, backtrack to the latest decision
      point, make a new decision and continue.
6. Check whether there is any value required at present state
   lines. If not, a test sequence is found. Fault simulate the new
   sequence and then go to Step 2.
   Otherwise, check
   a) whether the state requirement has been visited before or
   b) whether it has been identified as an unjustifiable state
   If yes, backtrack. Otherwise, assign the state requirement as
   the next state of the current time-frame and goto Step 5.
```

Fig. 11. The BACK algorithm.

CC_f^1 (s) and CC_f^0 (s) for every signal s are computed after the fault is injected. If an AND gate is injected, the extra primary input is hardwired to 1/0, and therefore CC_g^1 (*extra_PI*) is 0, CC_g^0 (*extra_PI*) is infinity, $CC_f^1(s)$ (*extra_PI*) is infinity, and CC_f^0(s) (*extra_PI*) is 0. The controllabilities of the signals in the fanout cone of the faulty signal can then be updated based upon the signal controllabilities of the fault-free circuits and the assigned controllabilities at the extra primary input. Once the fault-free and faulty controllabilities of all signals are available, drivabilities can be calculated. Each signal is associated with two drivability measures, *D_drive* and *$\bar{D}$_drive*, indicating the relative effort of propagating a D and a $\bar{D}$ from the fault site to the signal. For a signal s not reachable from the fault site, both *D_drive(s)* and *$\bar{D}$_drive(s)* are infinity. For the extra primary input introduced for fault injection, if the hardwired value is 1/0 (0/1), the *D_ drive* *($\bar{D}$_drive)* is set to 0 and the *$\bar{D}$_ ($\bar{D}$_drive)* is set to infinity. The drivabilities of signals at the fanout of the added gate can then be calculated. For example, consider a 3-input AND gate G with inputs A, B, and C. To produce a D at G requires that one of the inputs has a D and the other two inputs have value 1 *in the fault-free circuit* and *D_drive(G)* can be defined as the minimum among $\{D_drive(A) + CC_g^1(B) + CC_g^1(C)\}$, $\{D_drive(B) + CC_g^1(A) + CC_g^1(C)\}$, and $\{D_drive(C) + CC_g^1(A) + CC_g^1(B)\}$. Similarly, to propagate a $\bar{D}$ from an input of an AND gate to its output, the other inputs must have value 1 in the *faulty* circuit (In this case, the on-input has a controlling value 0 for the fault-free circuit and thus there is no value requirement at other inputs for the fault-free circuit). Therefore, *$\bar{D}$_drive(G)* is defined as the minimum among $\{\bar{D}_drive(A) + CC_f^1(B) +$

CC_f^1 $(C)\}$, $\{\bar{D}_drive(B)$ + CC_f^1 (A) + CC_f^1 $(C)\}$, and $\{\bar{D}_drive$ (C) + CC_f^1 (A) + CC_f^1 $(B)\}$.

To justify a D ($\bar{D}$) at the output of a gate during backward justification, the test generator selects the input of the smallest *D_drive* ($\bar{D}$*_drive)* as the D-input. In this drivability guidance, sensitization paths will be created implicitly and efficiently. The BACK algorithm is summarized in Figure 11. Some further speedup techniques can be found in several other recent developments [Niermann and Patel 1991; Schulz and Auth 1989; Kelsey et al. 1993; Lee and Reddy 1991; Glaeser and Vierhaus 1995].

HITEC. HITEC [Niermann and Patel 1991] employs several new techniques to improve the performance of test generation. Even though it uses both forward and reverse time processing, it clearly divides the test generation process into two phases. The first is the forward time processing phase in which the fault is activated and propagated to a primary output. The second phase is the justification of the initial state determined in the first phase using the reverse time processing. Due to the use of the forward time processing for fault propagation, several efficient techniques, such as the use of dominators, unique sensitization, and mandatory assignments [Fujiwara and Shimono 1983; Kirkland and Mercer 1987; Schulz et al. 1988; Schulz and Auth 1988] used in combinational ATPG can be extended and applied in phase 1. In the reverse time processing algorithms, such techniques are of no use. Also, no drivability is needed for the fault propagation phase, which further saves some computing time.

FASTEST. FATEST [Kelsey et al. 1993] uses only forward time processing and uses PODEM [Goel 1981] as the underlying test generation algorithm. For a given fault, FATEST first attempts to estimate the total number of time-frames required for detecting the fault and also estimate at which time-frame the fault is activated. The estimation is based on SCOAP-like [Goldstein 1979] controllability and observability measures. An iterative array model with the estimated number of time-frames is then constructed. The present state lines of the very first time-frame have an unknown value and cannot be assigned to either binary value. A PODEM-like algorithm [Goel 1981] is employed where the initial objective is to activate the target fault at the estimated time-frame. After an initial objective has been determined, it backtraces starting from the line of the initial objective until it reaches an unassigned primary input or a present state line in the first time-frame. For the latter case, backtracking is performed immediately. This process is very similar to the PODEM algorithm except that the process now works as a circuit model with multiple time-frames. If the algorithm fails to find a test within the number of time-frames currently in the iterative array, the number of time-frames is increased and test generation is attempted again based on the new iterative array.

Compared with the reverse time processing algorithms, the main advantage of the forward time processing algorithm is that it will not waste time justifying unreachable states and usually generates a shorter justification

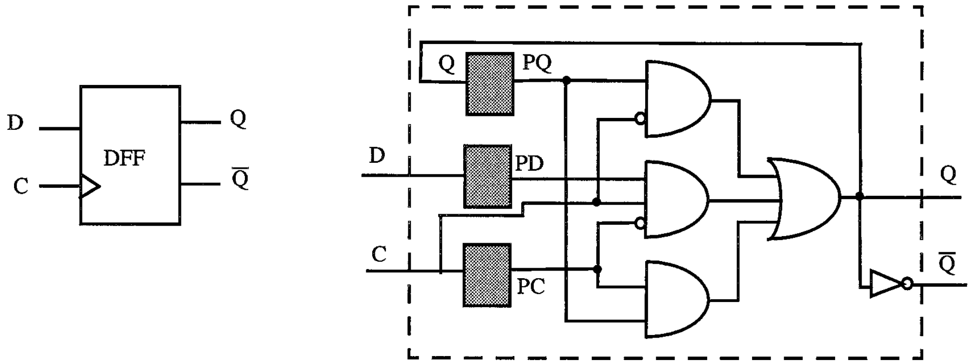

Fig. 12. The model of a positive-edge-triggered flip-flop.

sequence for bringing the circuit to a hard-to-reach state. For circuits with a large number of unreachable states or hard-to-reach states, the reverse time processing algorithms may spend much time in proving the unreachable states are unreachable or generating an unduly long sequence to bring the circuit to a hard-to-reach state. However, the forward time processing algorithm requires a good estimate of the total number of time-frames and the time-frame for activating each target fault. If the estimation is not accurate, the test generator may waste much effort in the smaller-than-necessary iterative array model.

2.1 Some Practical Issues

Test generation for multiple-clock and/or multiple-phase designs. The examples shown above are all single-clock and single-phase designs. The clock signal is not part of the primary input signals of the array model and a clock pulse needs to be applied between any two time-frames. To handle multiple-clock and/or multiple-phase designs, one simple solution is to treat the clock signals as part of the primary input signals and let the ATPG algorithm automatically figure out the required values at the clock signals. This solution requires converting an edge-triggered flip-flop into a model that consists of three delay elements [Cheng 1989]. For example, the 3-delay-element model for a positive-edge-triggered D flip-flop is shown in Figure 12. The three delay elements (shown in shaded boxes) store the previous value at the data input signal D, the previous value at the clock input signal C, and the previous state of the flip-flop Q, named a PD, PC, and PQ, respectively. It can be verified that this model correctly represents the behavior of a flip-flop. For example, if a sequence of (01) is applied at C, that is, C = 1 and PC = 0, then Q will be equal to PD. For a negative-edge-triggered D flip-flop, the model is almost the same except that an inverter is added before input signal C. After all flip-flops in the given circuit are replaced by their corresponding 3-delay-element models, the expanded circuit model consists of only delay elements. The iterative array model can then be used for the expanded model. The sequence generated already

specifies the required values at the clock signals and no additional clock sequence needs to be added between time-frames.

Example 3. Let's consider a simple circuit that consists of only one positive-edge-triggered D flip-flop and use the 3-delay-element model to find a test for the output Q stuck-at-0 fault. The circuit has two inputs D and C, connected to the data input and the clock input of the flip-flop, and one output Q. There are three possible cubes to set Q to 1 for activating the fault: {C = 0, PQ = 1}, {C = 1, PC = 0, PD = 1}, and {PC = 1, PQ = 1}. The first and the third cubes require PQ = 1 and in turn require Q = 1 again in the previous time-frame. This is a state repetition and therefore these two cubes need not be further explored. The state requirement of the second cube, PC = 0 and PD = 1, can be easily justified in the previous time-frame by setting C to 0 and D to 1. Therefore, we can conclude that the test is vector (CD = 01) followed by vector (CD = 1X), which is indeed the correct test.

Test generation for circuits with combinational loops. For circuits with some combinational feedback loops (i.e., loops that do not contain any flip-flops), the iterative array model can still be applied. This extension has been discussed in Chappell [1974] and Breuer [1974]. A set of feedback connections whose removal will eliminate all combinational feedback loops is first identified. At each of these combinational feedback lines, a *pseudo* delay element is inserted to create a new circuit model for test generation. The iterative array model is then constructed for the modified circuit which consists of two types of delay elements: regular delay elements (that are either D flip-flops in a single-clock/single-phase design or the ones in the 3-delay-element model discussed in the last paragraph) and the pseudo delay elements. In the iterative array model, the inputs to a time-frame include the primary inputs, the present state lines, and the outputs of the pseudo delay elements (called pseudo present state lines PPS). Similarly, the outputs of a time-frame consist of the primary outputs, the next state lines, and the inputs of the pseudo delay elements (called pseudo next state lines PNS). During test generation, additional constraints are imposed. For the time-frames in which at least one PPS/PNS line pair (associated with a pseudo delay element) does not have consistent binary values, each PS/NS line pair (associated with a delay element) is forced to have the same binary value and the primary inputs are not allowed to change values. If a PPS/PNS line pair does not have consistent binary values, the actual value at the corresponding combinational feedback line has not stabilized yet, so the clock should not be applied and the primary input vector should not change. By imposing such constraints, the iterative array will contain some fine-grained time-frames for consistent justification for the combinational feedback lines and some coarse-grained time-frames for value justification and fault propagation across the flip-flops.

Efficient identification of undetectable faults. Sequential ATPG is a complex search process. In principle, test generators are tuned to perform a

depth-first search such that a test sequence can be found as short and early as possible. A test generator will report a fault as undetectable after implicitly exhausting the search space. Because the search strategy is optimized for finding a test (instead of determining whether a fault is detectable or not), the undetectable faults are hard to identify and the test generators usually spend a significant fraction of total computation time on these undetectable faults. To limit the computational resource allocated to each fault, a test generator usually sets a limit on the time budget and a limit on the maximum number of time-frames for each target fault. For undetectable faults in larger circuits, the CPU time limit is usually reached before the search space is exhausted and they are usually classified as aborted faults instead of undetectable faults. Therefore, it may be worthwhile to add a preprocessing phase before ATPG which is dedicated to identifying undetectable faults and employs a different search strategy.

Several approaches suggest ways to identify a subset of undetectable faults based on efficient procedures [Cheng 1993; Agrawal and Chakradhar 1995; Pomeranz and Reddy 1994; Iyer and Abramovici 1994]. For example, a fault which is undetectable under the assumption that all state lines are fully controllable and observable (i.e., combinationally undetectable) is also a sequentially undetectable fault. Such an undetectable fault can be much more efficiently identified by a combinational circuit test generator. Note that such a fault may not be identified as an undetectable fault by a sequential circuit test generator if just one time-frame is allowed. For example, if the BACK algorithm is used for a combinationally undetectable fault and a fault effect (D or $\bar{D}$) is assigned to a primary output, it is still possible that the fault effect may be back propagated to a present state line of this time-frame and the fault cannot be identified as undetectable using just one-time-frame sequential ATPG. A procedure for identifying a special class of undetectable faults, called feedback-free sequentially undetectable faults, is suggested in Cheng 1993. The procedure first converts a sequential circuit into a feedback-free model by assuming a set of feedback lines as fully controllable and observable. A fault which is undetectable under this model will also be sequentially undetectable. The method proposed in Agrawal and Chakradhar [1995] uses a combinational circuit test generator to target faults in the iterative array model with a pre-specified, small number of time-frames. It assumes that the next-state lines of the last (i.e., rightmost) time-frames are fully observable and the present state lines of the first (i.e., leftmost) time-frames are fully controllable. Two different procedures are suggested to identify a subset of sequentially undetectable faults: one assumes that only the last time-frame of the array model has the target fault and the other time-frames are all fault-free; the other assumes that every time-frame in the array model has the target fault. In other words, the first procedure targets a single-stuck-at fault in the array model and the second procedure targets a multiple-stuck-at fault. It has been shown [Agrawal and Chakradhar 1995; Pomeranz and Reddy 1994] that a combinationally undetectable fault in the iterative array model identified by either one of the two procedures corresponds to a sequentially

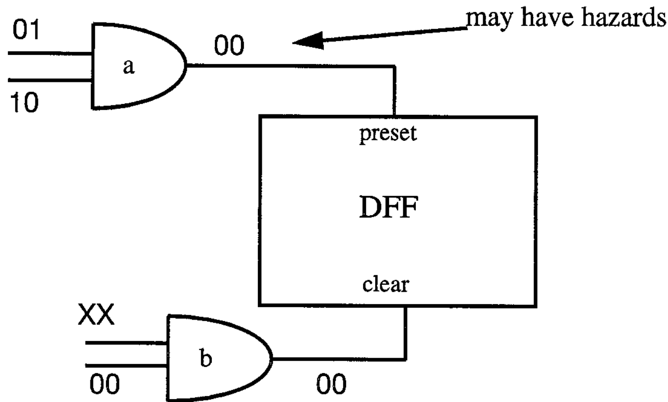

Fig. 13. Nonrobust test at asynchronous preset/clear lines.

undetectable fault in the original circuit. Therefore, such techniques will allow identification of a subset of sequentially undetectable faults using more efficient combinational techniques. In Pomeranz and Reddy [1994], a more detailed characterization of these undetectable faults in terms of their classification with respect to redundancy is given. The relationship between undetectability and redundancy will be discussed in more details in Section 2.2. In Iyer and Abramovici [1994], an efficient algorithm based purely on implication is proposed. The algorithm relies on identifying illegal value combinations (also called *conflicts*) on a subset of signals in the circuit. Faults for which these illegal value combinations are necessary for detection are then found as sequentially undetectable.

Test efficiency. The fault coverage figures achieved by the test sequence produced by a test generator may not be a fair indicator of the test generator's performance. Determining the set of undetectable faults is also an objective of test generation. Therefore, the *test efficiency*, defined as the ratio of the number of detected faults plus the number of identified undetectable faults to the total number of faults, is usually used to evaluate the efficiency of a test generator.

Robust Tests for Asynchronous Preset/Clear Lines and for Busses

Special attention needs to be paid for asynchronous preset/clear lines. No hazards or glitches are allowed at these signals; otherwise the test may be invalidated. This problem has been discussed in detail in Breuer and Harrison [1974].

Example 4. Consider the example shown in Figure 13. The test may create hazards at the output of gate a and thus may accidentally preset the flip-flop. Stable, inactive signal values, such as the one at output of gate b, must be maintained at asynchronous clear and preset lines. q

A circuit may have busses that do not use fully decoded enables. Even if the designer is confident that during system operation, no vector creates a

state that could cause bus contention or bus floating, it cannot be guaranteed that the sequences generated by an ATPG tool would not cause bus problems. The ATPG tools need to make sure that the sequence generated will not cause bus contention or leave the bus floating, even for those busses not in the fanin or fanout cones of the target fault. This constraint can be implemented by adding some fictitious logic to the fanin of the bus enables and data signals to make sure that one or more bus enables are active. If more than one enables are active, the fictitious logic will force the corresponding bus data to be consistent.

Complexity of Sequential ATPG

For the topological analysis based approaches, the complexity can be roughly measured by the average number of vectors (therefore, time-frames) required for detecting a fault in the given network. In general, the average number of vectors for detecting a fault is highly related to the structure of the sequential circuit and attributes, such as sequential depth (the maximum number of flip-flops encountered in any path from a primary input to a primary output), number of sequential cycles (a sequential cycle exists if a flip-flop's data input is in the fanout cone of the flip-flop's output), and the maximum length of any sequential cycles (the length of a cycle is defined as the number of flip-flops in the feedback loop). The presence of sequential cycles increases the temporal correlations of values on specific nodes and costs more time-frames to justify specific values at internal nodes. It has been shown [Miczo 1986; Cheng and Agrawal 1989; Gupta and Breuer 1990] that the computational complexity of test generation for a cycle-free sequential circuit is not too much higher than that for a combinational circuit. For a cycle-free circuit with a sequential depth D, any detectable single-stuck-at fault can be detected by a sequence of length $D + 1$. Therefore, for such circuits, there is a small upper-bound on the number of time-frames for each test generation attempt and a high fault coverage can usually be achieved. Empirical studies have also shown that the average number of time-frames is somewhat linearly proportional to the sequential depth of a circuit and is exponential in terms of the maximum cycle length [Miczo 1986; Cheng and Agrawal 1989].

ATPG may waste a substantial amount of runtime in fruitless justification of invalid states (defined as the states that cannot be reached from the reset state). The computational complexity caused by such fruitless efforts is not reflected in the above mentioned circuit attributes such as sequential depth and sequential cycles. Recently, it has been pointed out [Marchok et al. 1995] that *density of encoding*, defined as the ratio of the number of valid states to the total number of states, is a key indicator of topological-analysis-based sequential ATPG. The lower the ratio, the more the invalid state, and the higher the probability of a sequential ATPG wasting time in justifying them during test generation. It has been shown [Marchok et al. 1995] that ATPG spent significantly longer CPU time and achieved a lower fault coverage for a retimed version of a sequential circuit that has a much lower density of encoding than the original circuit.

2.2 Undetectability and Redundancy

For combinational circuits or full-scan sequential circuits, a fault is called undetectable if no input sequence can produce a fault effect (D or $\bar{D}$) at any primary output, and a fault is called redundant if the presence of the fault does not change the input/output behavior of the circuit. The detectability is associated with a test generation procedure while redundancy is associated with functional specification of a design. A fault is combinationally redundant if it is reported as undetectable by a *complete* combinational test generator. The definitions of detectability and redundancy for (non-scan) sequential circuits are much more complicated [Abramovici et al. 1990; Cheng 1993; Pomeranz and Reddy 1993], and these two properties (redundancy and undetectability of stuck-at faults) are no longer equivalent [Abramovici et al. 1990; Cheng 1993; Pomeranz and Reddy 1993]. It is pointed out in Pomeranz and Reddy [1993] that undetectability can be precisely defined only if a test strategy is specified, and redundancy cannot be defined unless the operational mode of the circuit is known. Formal and precise definitions of undetectability with respect to four different test strategies, namely *full-scan*, *reset*, *multiple observation time*, and *single observation time*, are given, and redundancies with respect to three different circuit operational modes, namely *reset*, *synchronization*, and *non-synchronization* are also defined in Pomeranz and Reddy [1993]. A fault is called undetectable under full-scan if it is combinationally undetectable [DeVadas 1990]. In the case where hardware reset is available, a fault is said to be undetectable under the reset strategy if no input sequence exists such that the output response of the fault-free circuit is different from the response of the faulty circuit, both starting from their reset states. In the case where hardware reset is not available, there are two different test strategies: the *multiple observation time* (MOT) strategy and the *single observation time* (SOT) strategy. Under the SOT strategy, a sequence detects a fault only if a fault effect appears at the same primary output O_i and at the same vector V_j for all power-up initial state-pairs of the fault-free and faulty circuits (O_i could be any primary output and V_j could be any vector in the sequence). Most gate-level test generators and all ATPG algorithms mentioned in Section 2 assume the SOT test strategy. Under the MOT strategy, a fault can be detected by multiple input sequences—each input sequence produces a fault effect at some primary output for a subset of power-up initial state-pairs and the union of the subsets covers all possible power-up initial state-pairs (for a n-flip-flop circuit, there are 2^{2n} power-up initial state-pairs). Under the MOT strategy, it is also possible to detect a fault using a *single test sequence* for which fault effects appear at different primary outputs and/or different vectors for different power-up initial state-pairs.

Example 5. Figure 14 shows an undetectable fault under the SOT test strategy [Cheng 1993]. Line C stuck-at-0 fault prevents the flip-flop from initialization. Since no input sequence can set the flip-flop to a known binary value in the faulty machine, no input sequence can produce a D (1/0)

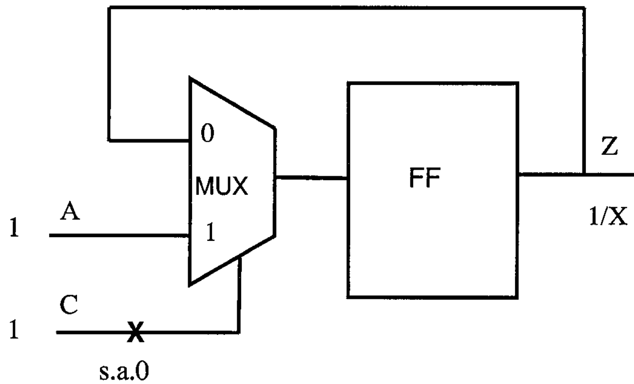

Fig. 14. A fault that is SOT undetectable but MOT detectable.

or $\bar{D}$ (0/1) at the primary output. Note that this fault is definitely not redundant—fixing line *C* to constant 0 completely changes the functional behavior of the circuit. The fault can actually be thoroughly tested by test sequence AC = (11, 01). If the initial state of the flip-flop is X/0 (i.e., the initial state of the fault-free circuit is unknown and that of the faulty circuit is 0), the output response of the first vector will be 1/0 and the fault is detected. If the initial state is X/1, the output response will be 1/1 for the first vector and 0/1 for the second vector and the fault is also detected. Since the union of X/0 and X/1 covers all possible combinations of initial fault-free and faulty state-pairs, this sequence detects the fault under the MOT test strategy. Note that it cannot be determined a priori which vector will produce a fault effect at PO for this fault and, therefore, the outputs must be observed for both vectors. We can then conclude that this fault is undetectable under the SOT strategy and detectable under the MOT strategy.

An algorithm for generating tests under the MOT strategy has been proposed in Pomeranz and Reddy [1992]. The advantages of the MOT test strategy include the following: (1) There are faults that are undetectable under the SOT strategy but detectable under the MOT strategy. Therefore, there will be more detectable faults if MOT is used. (2) Even if a fault is detectable under the SOT strategy, it may still be advantageous to use the MOT strategy because the MOT test is, in general, shorter than the SOT test. Under the SOT strategy, the test sequence is preceded by an initialization sequence that brings both fault-free and faulty circuits to a known state. Such a sequence could be long. Full initialization may not be necessary for the MOT strategy. However, under the MOT strategy, the test generation process is more complex and the tester needs to store not just one but several fault-free responses of the test vectors and compare the response of each device under test against them. Existing testers do not provide this feature yet.

In Pomeranz and Reddy [1993], sequential redundancy is defined with respect to three possible modes of operation: (1) reset mode, in which the circuit is reset to a specific initial state at the beginning of operation, (2)

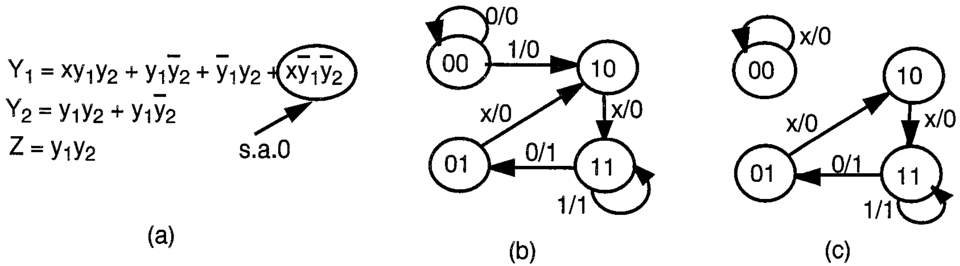

Fig. 15. An example of a partially detectable fault. (a) Boolean functions and the stuck-at-fault; (b) the fault-free state graph; (c) the faulty state graph.

synchronization mode, in which the operation starts with a specific initialization sequence, and (3) no-synchronization mode, in which the operation starts from the state in which the circuit happens to be. A fault is redundant under a given mode of operation if, for all possible input sequences applicable under the given mode of operation, the fault does not change the expected output responses. For circuits with a hardware reset, a fault is undetectable under the reset test strategy if and only if the fault is redundant under the reset mode of operation. The notion of partial tests introduced in Pomeranz and Reddy [1993] is useful in understanding the relationship between undetectability and redundancy under the synchronization and the no-synchronization modes of operation. A partial test is an input sequence such that for *at least one initial state* of the faulty circuit, the output response of the faulty circuit is different from the response of the fault-free circuit [Pomeranz and Reddy 1993]. A fault is called partially detectable if at least one partial test exists. It is shown [Cheng 1993; Pomeranz and Reddy 1993] that a fault is redundant if and only if it is not partially detectable. A modification to the sequential circuit test generation algorithm for identifying partially detectable faults can be found in Cheng 1993.

Example 6. Consider the two-level circuit shown in Figure 15(a). The target fault is the output stuck-at-0 fault of the AND gate ($x\bar{y}_1\bar{y}_2$). Figure 15(b) and Figure 15(c) show the state graphs of the fault-free and faulty circuits. Sequence {x = 011} will initialize the fault-free circuit to state $y_1y_2 = 11$ but the faulty circuit cannot be initialized because state $y_1y_2 = 00$ is a sink state. The fault-free response at output Z for input sequence {x = 0110} is {Z = XXX1}. If the faulty circuit initial state is $y_1y_2 = 00$, the faulty response at Z for the same input sequence will be {Z = 0000} and the target fault can be detected by the sequence. However, if the faulty circuit initial state is y_1y_2 = 01, 10, or 11, the faulty response at Z will be {Z = XXX1} and the fault will not be detected. In summary, the fault is detectable if the power-up initial state of the faulty circuit is 00. Otherwise, it is not detectable. Therefore, the fault is partially detectable. q

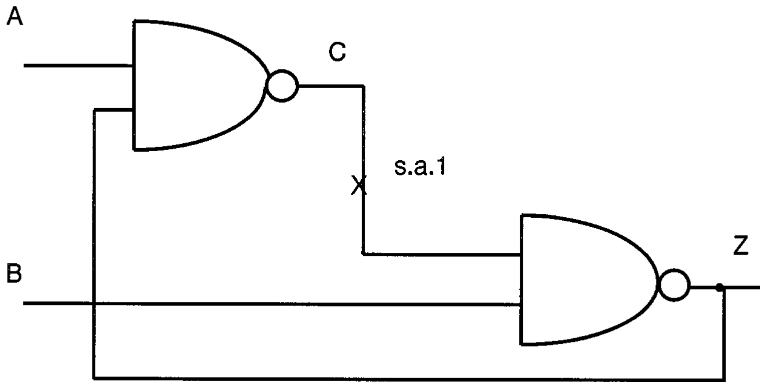

Fig. 16. A NAND latch.

2.3 Deficiencies

Theoretical limitation. Consider a 20-bit counter. It takes a sequence of 2^{19} vectors to detect the stuck-at-0 fault at the most significant bit. Therefore, this many time-frames would be required in the test generation process. Such a small but highly sequential circuit could hardly be handled by any gate-level ATPG tool. Sequential ATPG is by no means a complete solution for highly sequential circuits. However, sequential ATPG combined with design-for-testability techniques such as partial scan offers a complete solution. The relationship between sequential ATPG and partial scan will be discussed in Section 5.

Timing. Timing is not properly considered by approaches based on the iterative array model for circuits with combinational loops and for multiple-phase or multiple-clock designs, as discussed earlier. Even though we can construct an iterative array model for an asynchronous circuit and apply the path sensitization technique on the model to generate tests, the tests generated in such a manner may cause races and hazards.

Example 7. [Cheng and Agrawal 1989]. Figure 16 shows a cross-coupled NAND latch. The iterative array model of this latch constructed by cutting the feedback path is shown in Figure 17. The test shown in Figure 17 is vector (AB = 11) preceded by vector (AB = X0) where X denotes the don't care state. The value assigned to the unspecified signal may cause a problem. If we set X to 1, we get the desired test. But if X is set to 0, the test will cause *a race in the fault-free circuit*. Thus, tests generated by such procedures require special processing to avoid timing problems. The simplest way is to use a fault simulator that takes timing into account to verify the generated tests. *q*

Over-specification. Some array-model-based test generators may err in identifying undetectable faults [Cheng and Ma 1993]. In other words, these test generators may not find the test sequence for a detectable fault under a given test strategy, even allowed infinite run time, and furthermore may

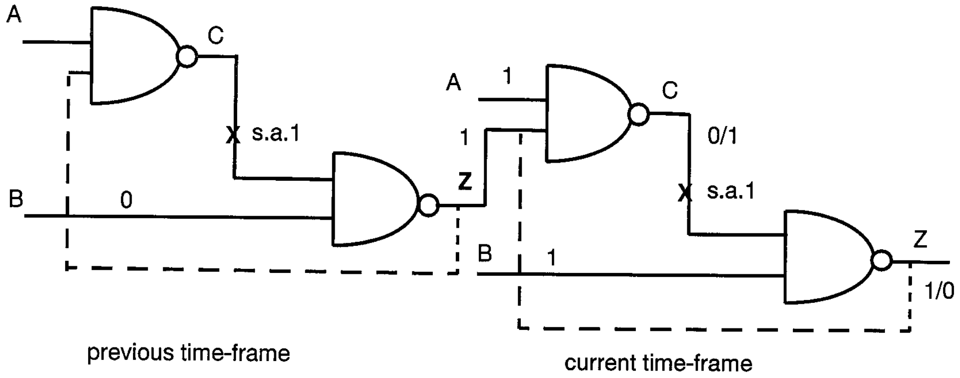

Fig. 17. Time-frame expansion for an asynchronous circuit.

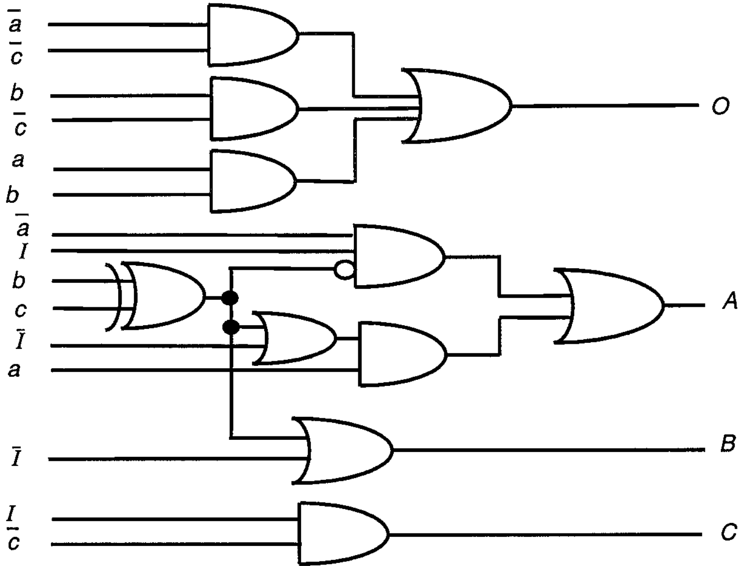

Fig. 18. An example illustrating the over-specification problem.

mistakenly claim it as undetectable. The main problem of these test generators is that the underlying combinational test generation algorithm, for example, PODEM [Goel 1981], may over-specify the requirements at the present state lines.

Example 8. Consider the circuit given in Figure 18 that has three flip-flops, one input (I), and one output (O). The next state lines are A, B and C and the corresponding present state lines are a, b, and c. We assume the single observation time test strategy. Note that flip-flop A is not initializable. Let us try to generate a test for the stuck-at-0 fault at output O. PODEM is used within each time-frame as the underlying combinational test generation algorithm. Let's consider only the last (i.e., the rightmost)

Table I. Combinational Tests of the Last Time-Frame Found by PODEM

	I	a	b	c
T_1	X	0	X	0
T_2	X	1	1	0
T_3	X	1	1	1

time-frame. Assuming that the order of assigning values at the present state lines at this time-frame during the search is c followed by a, followed by b, the complete set of combinational tests for this time-frame found by PODEM is shown in Table 1. Because flip-flop A is not initializable, none of these three required states (abc = 0X0, 110, 111) specified in the combinational tests can be justified. The test generator would conclude that the fault is undetectable under the SOT strategy. However, vector T_4: ($Iabc$ = XX10) that is jointly covered by tests T_1 and T_2 is also a combinational test but it is not generated by PODEM. The required state of T_4 (abc = X10) has an X value at the present state line a and can be justified by a single vector $I = 0$. Thus the stuck-at-0 fault at output O can be detected by the 2-vector test sequence (I = 0, I = X). q

The main problem here is that not all minimally-specified combinational tests are generated. It may be intuitive to resolve the problem by expanding the combinational tests to convert the over-specified present state lines into X's. This simple solution is to choose a present state line i whose value is specified (i.e., set to 0 or 1) and change it to a don't-care value X. If the new vector is still a combinational test (verified by simulation), a less specified test is found. This process continues until all present state lines are processed once. This simple heuristic may help but still does not guarantee finding all minimally specified tests. A necessary condition that the underlying combinational test generation algorithm must satisfy to be able to generate all minimally specified tests and, thus, ensure a correct sequential test generator (in terms of identifying undetectable faults) is given in Cheng and Ma [1993]. The D-algorithm [Roth 1966] satisfies the necessary condition while some of the well-known combinational algorithms such as PODEM violate this condition. Modifications of these algorithms to meet this condition were also suggested in Cheng and Ma [1993].

3. SIMULATION-BASED APPROACHES

Another class of gate-level approaches, called simulation-based approaches [Seshu and Freeman 1962; Cheng and Agrawal 1989; Saab et al. 1992; Rudnick and Patel 1994; Prinetto et al. 1994; Nitta et al. 1985; Rudnick et al. 1994; Srinivas and Patnaik 1993; Pomeranz and Reddy 1995; Corno et al. 1996], is enhanced from a logic simulator or a fault simulator. These approaches are combinations of simulation, and cost function guidance. The basic concept of these approaches is as follows: Suppose we wish to

generate a test for a given fault or set of faults. Based upon the previous tests, a new trial vector or a trial sequence is generated. Logic simulation or fault simulation is then performed for this trial vector/sequence. From the simulation result, a cost function that, in some way, determines how *far* the state of the circuit is from the required state for the detection of the fault(s), is computed. If the cost is reduced, the new trial vector/sequence will be included in the final test sequence; otherwise it will be discarded. A new trial vector is then generated and the process repeats until the fault is detected or the number of trial vectors reaches a pre-specified limit. The differences between different methods in this class of approaches are in (1) the ways to generate new trial vectors and (2) the cost functions for guiding the search. Most simulation-based methods consist of multiple phases, and different phases use different cost functions to achieve different objectives.

The first simulation-based method was proposed by Seshu and Freeman in 1962. A fault simulator is used to evaluate trial vectors, which are defined as those vectors that differ from the present vector in exactly one bit position. The trial vector that detects most faults will be accepted as the next test vector. Other trial vectors that detect any fault are saved in a stack to be used as bases for generating new trial vectors. If no trial vector detects any fault, then the algorithm selects one vector from the stack and uses it as the present vector. A set of trial vectors is then generated and the process continues until the stack is empty. A few heuristics have further been proposed in Seshu [1965]. For example, when the stack is empty and none of the trial vectors is acceptable, instead of terminating the process, a new vector is randomly generated and trial vectors are generated from the random vector.

Breuer proposed another simulation-based method in 1971. The trial vectors are simply random vectors. At each iteration, a fixed number of random vectors are generated and they are evaluated by a fault simulator. The best vector among the trial vectors is then selected as the next test vector. The cost function for evaluating each trial vector (V) is in the following form: $C(V) = aC_1 + b(C_2 - C_3)$ where C_1 is the number of new faults detected by V, C_2 is the number of faults whose fault effects were not present in any flip-flop before the application of V but are present in some flip-flop after the application of V, C3 is the number of faults whose fault effects were present in some flip-flop before the application of V but are not present in any flip-flop after the application of V, and a and b are just weighting constants.

SOFTG (Simulator-Oriented Fault Test Generator) [Snethen 1977] is a test generator that can be classified as simulation-based while it also uses the backtrace procedure of PODEM in finding the test. To ensure hazard-free tests, SOFTG generates a test sequence in which each vector differs from its previous vector in one bit and uses a simulator for all forward signal propagation. The backtrace procedure is used to find the best input bit to be flipped. The close interaction with the simulator allows SOFTG to effectively model the timing behavior of a sequential circuit such that the tests will not cause races or hazards in the fault-free circuit.

CONTEST [Cheng and Agrawal 1989] is a test generator enhanced from a concurrent fault simulator. CONTEST generates a new trial vector by changing a single bit in the last accepted vector. The test generation process is subdivided into three phases with a different cost function for each phase: (1) Initialization phase—to bring flip-flops in the circuit into known states irrespective of the starting state. The cost function of this phase is simply the number of uninitialized flip-flops. Initially, the cost may be equal to the number of flip-flops in the circuit. The goal is to reduce this cost to 0. (2) Concurrent fault detection phase—the objective is to generate tests effectively by targeting all undetected faults concurrently. The cost function of a fault is defined as the minimum distance of any fault effects created by this fault to any primary output. The distance is simply the number of logic gates on the path. The smaller the cost, the closer the fault is to being detected. When a fault is detected, its cost will be zero. If a fault is not activated, the cost is defined as infinity. In general, there are many undetected faults in phase 2. The costs of all undetected faults for the current vector and for the candidate trial vector needs to be computed. To determine whether to accept the candidate vector or reject it, two lists of cost functions instead of two numbers are compared. That is, the search for tests is guided by a group of faults instead of a single target fault. Based on this principle, a simple cost function for a trial vector can be defined as the sum of, say, 10% of the lowest cost undetected faults. The concurrent phase stops when no single bit changes in the current vector produce cost reduction. This will normally happen when a small number of faults is left in the target set. (3) Single fault detection phase—test vectors are generated for a single target fault. The cost function is based on a SCOAP-like testability measure [Goldstein 1979]. For a target fault, *dynamic* testability measures that estimate the minimum number of primary inputs of the current vector that *must be changed* and the minimum number of *additional* vectors that must be applied for detecting the fault are used to measure the cost of a trial vector and to guide vector generation. Similar to phase 2, a new trial vector is generated based on a single bit change of the current vector.

3.1 Generating Trial Vectors by Genetic Algorithm

CRIS [Saab et al. 1992], another simulation-based test generator, uses two cost functions that are based on (1) the distance of the fault effects to the primary outputs and (2) the distribution of the switching activity of the previously generated tests. The first cost function is similar to the one used in phase 2 of CONTEST. The second cost function is used to select vectors that would create more switching activity for the portion of the circuit that has a low level of switching activity for the previously generated tests. Increasing the switching activity in an inactive subcircuit would increase the probability of detecting faults in that subcircuit. CRIS uses the genetic algorithms [Goldberg 1989] to generate new trial vectors. The new trial vectors are generated by some basic evolutionary operators of genetic

algorithms such as *reproduction* (copying potentially useful candidate vectors and sequences), *mutation* (flipping bits in a vector), *splicing* of vectors (producing a new vector using substrings from two other vectors), or sequences (producing a new sequence from subsequences of existing sequences). The information collected during simulation (such as activity of internal nodes and nodes where the fault effects were blocked) is used to identify bits for mutation or the vectors and sequences for splicing. GATTO [Prinetto et al. 1994] is another simulation-based method using genetic algorithms. Similar to CRIS, the cost function of GATTO is to maximize the circuit activity.

GATEST [Rudnick et al. 1994] uses genetic algorithms to generate new trial vectors as well as new trial sequences. The cost function (called fitness function in genetic algorithms) of each trial vector or sequence is evaluated by a fault simulator. The main objective of the fitness function is to maximize the number of faults detected and the number of fault effects propagated to flip-flops. GATEST consists of four phases: (1) Phase 1 (the initialization phase)—The fitness function consists of two terms. The first is the count of initialized flip-flops, which is the same as the phase 1 cost function used by CONTEST. The second term is the fraction of flip-flops changing values after the application of the vector under evaluation. The second term is used to differentiate vectors that cause the same number of flip-flops to be initialized. The vector that causes more state variables to change values is preferred. When all flip-flops are initialized, the program enters phase 2. (2) Phase 2—The fitness function is the sum of two terms: (i) the number of faults it detects and (ii) the number of faults whose fault effects propagated to flip-flops are divided by the product of the total number of faults and the total number of flip-flops. Again, the second term is a fraction and is used to differentiate vectors that detect the same number of faults. Once a test vector is generated that detects no additional faults, the program enters the next phase. (3) Phase 3—The fitness function consists of three terms and the first two terms are identical to those of phase 2. The third term is the total number of good and faulty circuit events divided by the product of the total number of nodes and the total number of faults. This cost function is used to select the vectors that create high good and faulty circuit activity levels. In phase 3, the vectors normally do not detect any fault (i.e., the first term is zero) and the objective is to change the circuit to a new state such that additional faults can be detected. When a fault is detected in phase 3, the program goes back to phase 2 and the process repeats. If the number of vectors generated in phase 3 exceeds a limit, the program enters the final phase. (4) Phase 4—The fitness function is evaluated for trial *sequences* instead of individual trial *vectors*. The fitness function of a trial sequence is very similar to that of phase 2 for a trial vector, except that the second term is further divided by the trial sequence length.

It has been pointed out in Rudnick et al. [1994] that several genetic algorithm parameters are important in achieving good results. The GA parameters include the population size (i.e., the number of vectors used to

generate new trial vectors), the number of generations for evolving the vectors, and the crossover and mutation probabilities in evolving the vectors.

The simulation-based approaches have several advantages: (1) Logic simulation and fault simulation deal with circuit delays in a very natural way, and therefore even highly asynchronous sequential circuits can be handled. It can be assured that the sequence generated by such approaches will not cause hazards/races in the asynchronous circuits. (2) Because a fault simulator can accept a transistor-level netlist, it can handle transistor-level fault models. (3) Fault simulators, in general, require less CPU run time than array-model-based approaches.

The major disadvantages of the simulation-based approaches are: (1) Unlike the array-model-based approaches, these methods cannot identify undetectable faults. (2) They may also fail to generate tests for certain *hard-to-activate* faults. The guidance provided by the cost functions of these faults is usually very weak. (3) The test sequence generated tends to be longer than that generated by the array-model-based approach.

3.2 Hybrid Approaches

A number of hybrid methods have recently been proposed that combine the simulation-based technique with topological-analysis-based techniques [Saab et al. 1994; Rudnick and Patel 1995; Hsiao et al. 1996]. CRIS-hybrid [Saab et al. 1994] combines CRIS [Rudnick et al. 1994] with a iterative-array-model-based ATPG in a loose way. When CRIS generates a fixed number of test vectors without improving the fault coverage, array-model-based test generation is applied to get a set of vectors to be added to both the final test sequence and also to the current population to be used for trial vector generation for further call to CRIS. The array-model-based test generation is also used to identify untestable faults. The results show that this hybrid method results in test sequences of a better quality in terms of both test length and fault coverage as compared to CRIS. GA-HITEC [Rudnick and Patel 1995], another hybrid test generator, uses topological algorithms for fault excitation and fault propagation and uses genetic algorithms for state justification. If the genetic approach is not successful, topological-analysis-based procedures are further used for state justification. Further improvement to GA-HITEC, to closely integrated HITEC with a GA-based test generator, is given in Hsiao et al. [1996].

4. APPROACHES ASSUMING A KNOWN RESET STATE

To avoid the generation of an initialization sequence, a class of approaches assumes a known initial state. This assumption is valid for controllers that usually have a hardware reset (i.e., there is an external reset signal and the memory elements are implemented by resettable flip-flops). Approaches like STALLION [Ma et al. 1988], STEED [Ghosh et al. 1991], and VERITAS [Cho et al. 1993] belong to this category.

STALLION [Ma et al. 1988] first extracts the state transition graph (STG) for the fault-free circuit. For a given fault, it finds an activation state S and a fault propagation sequence T that will propagate the fault effect to a primary output. This process is based on PODEM and the iterative array model. There is no backward state justification in this step. Using the state transition graph, it then finds a state transfer sequence T_0 from the initial state S_0 to the activation state S. Because the derivation of the state transfer sequence is based on the state graph of the fault-free circuit, the sequence may be corrupted by the fault and, thus, may not bring the faulty circuit into the required state S. Therefore, fault simulation for the concatenated sequence $T_0 - T$ is required. If the concatenated sequence is not a valid test, an alternative transfer sequence or propagation sequence will be generated. STALLION performs well for controllers for which the state transition graph can be extracted easily. However, the extraction of STG is not feasible for large circuits. To overcome this deficiency, STALLION constructs a partial STG only. If the required transfer sequence cannot be derived from the partial STG, the partial STG is then dynamically augmented.

STEED [Ghosh et al. 1991] is an improvement of STALLION. Instead of extracting the complete or partial state transition graph, it generates ON-set and OFF-set for each primary output and each next state line for the fault-free circuit during the preprocessing phase. The ON-set (OFF-set) of a signal is the complete set of cubes (in terms of the primary inputs and the present state lines) that produces a logic 1 (logic 0) at a signal. The ON-sets and OFF-sets of the primary outputs and next state lines can be generated using a modified PODEM algorithm. For a given fault, PODEM is used to generate one combinational test. The state transfer sequence and fault propagation sequence are constructed by intersecting the proper ON/OFF-sets. In general, an ON/OFF-set is a more compact representation than the state transition graph. Therefore, STEED can handle larger circuits than STALLION. STEED shows good performance for circuits that have relatively small ON/OFF sets. However, generating, storing, and intersecting the ON/OFF-sets are very expensive (in terms of both CPU time and memory) for certain functions, for example, parity trees, and therefore STEED may have difficulties generating tests for circuits containing such functions. Also, like STALLION, the transfer and fault propagation sequences derived from the ON/OFF sets of the fault-free circuit may not be valid for the faulty circuit and therefore need to be verified by a fault simulator.

VERITAS [Cho et al. 1993] is a BDD-based test generator which uses the Binary Decision Diagram (BDD) to represent the state transition relations as well as sets of states. In the preprocessing phase, a state enumeration algorithm based on such BDD representations is used to find the set of states that are reachable from the reset state and the corresponding shortest transfer sequence for each of the reachable states. In the test generation phase, similar to STEED, a combinational test is first generated. The state transfer sequence to drive the machine into the activation

state is readily available from the data derived from a reachability analysis done in the preprocessing phase. Due to the advances in BDD representation, construction, and manipulation, VERITAS in general achieves better performance than STEED.

In addition to the assumption of a known reset state, another common principle used by the above three approaches is to incorporate a preprocessing phase to compute (explicitly or implicitly) the state transition information. Such information could be used during test generation to save some repeated and unnecessary state justification effort. However, for large designs where the state space is huge, such preprocessing could be an overkill. For example, complete reachability analysis used in the preprocessing phase of VERITAS typically fails (due to memory explosion) for designs with several hundreds of flip-flops. Either using partial reachability analysis or simply performing state justification on demand during test generation is a necessary modification for large designs.

4.1 Computing Reset Sequences

For circuits with an unknown reset state, it is possible to find a synchronizing sequence (or called reset sequence) to drive the circuit into a fixed known state and then, starting from that state, to use one of the above mentioned test generators (STALLION, STEED, or VERITAS) for test generation. However, because the reset sequence may be corrupted by some faults, the derived test sequence needs to be validated by a fault simulator.

Several approaches are recently proposed for determining whether a reset sequence exists and for efficiently computing short reset sequences. Pixley and others [Pixley and Beihl 1991] show that the problem of deciding whether a design is resettable can be solved without actually finding a reset sequence. If a reset sequence exists, the method calculates the reset sequence heuristically by implicit enumeration of states of the product machine of the design (the product machine of the design is a disjoint union of two copies of the design sharing their primary inputs) [Cho et al. 1993; Pixley et al. 1994]. This procedure is based on the Fundamental Alignment Theory developed in Pixley [1990]. In Rho et al. [1993], a technique for generating minimum length reset sequences is proposed. The method is based on BDD model of the iterative array model of the sequential circuit. It implicitly calculates all possible minimum length sequences and corresponding reset states by deciding the satisfiability of a boolean formula derived from the next state functions. The main disadvantage of this method is that the exhaustive nature of this method may not be suitable for large designs. Further improvement to this method is reported in Keim et al. [1996]. In Wehbeh and Saab [1994], a partitioning technique is used for handling larger designs. The circuit is decomposed into multiple smaller components and the initializability techniques are applied starting from the first level partitions toward the higher levels.

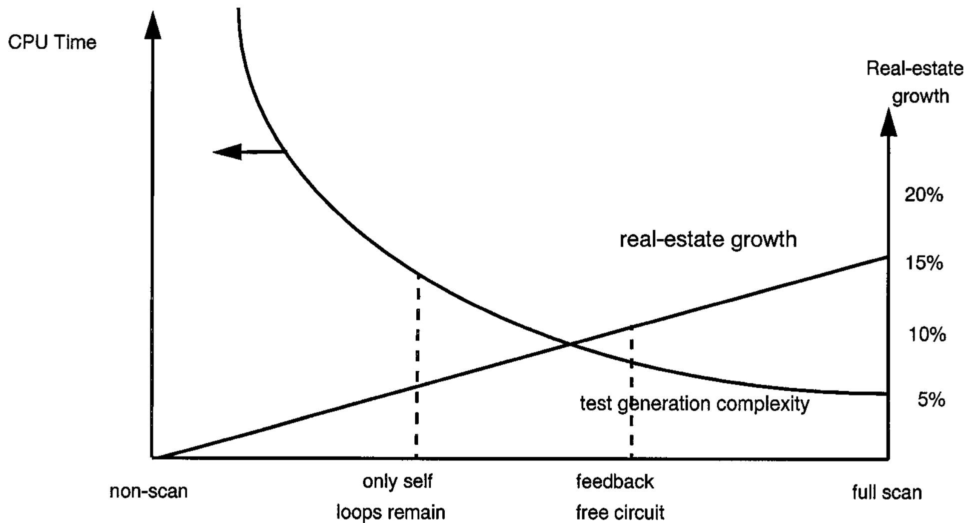

Fig. 19. Trade-off of area overhead vs. test generation effort.

5. PARTIAL SCAN AND SEQUENTIAL CIRCUIT TEST GENERATION

Due to its high complexity, sequential circuit test generation alone is still not a complete solution to testing sequential circuits. For highly sequential circuits, design for testability is still required. Many recently proposed partial-scan techniques (e.g., Cheng and Agrawal 1990; Gupta and Breuer 1990) can automatically identify a set of memory elements that would cause problems for the test generators and select these flip-flops for scan. The first-order criterion for selecting scan flip-flops is to break all cycles such that the resulting partial-scan circuit is feedback-free. As discussed earlier, the test generation complexity for feedback-free circuits is relatively low and test generators can usually achieve high fault coverages for such circuits. It has also been shown empirically [Cheng and Agrawal 1990] that cycles of length greater than 1 pose a greater problem for test generation than cycles of length 1 (self-loops). Therefore, if we cannot scan all the flip-flops necessary for breaking all cycles due to the area and/or the performance constraints, the longer cycles should be broken first. Figure 19 shows the trade-off curve of the area overhead versus the test generation complexity. This figure only illustrates the general trend. The actual trade-off curves for different circuits may vary. The horizontal axis represents the percentage of scan. Four cases are explicitly marked in the axis: (1) The non-scan case. The test generation complexity for such circuits is generally very high and, even for a medium size circuit, a high fault coverage can seldom be achieved. (2) The case that all loops of length greater than one are broken (by scanning a proper subset of flip-flops). For many medium size benchmark circuits, the computational complexity drops to a level where an existing test generator could produce high coverage tests. For larger benchmark circuits, some of the self-loops still need to be

broken by selecting more scan flip-flops. (3) The case that all loops, including self loops, are broken. (4) The full-scan case for which all flip-flops are scanned and a combinational test generator is used. The area overhead is roughly proportional to the percentage of scan and is estimated by the straight line in the graph. To date, most of the ASIC designs still use 100% full scan or a very high percentage of scan (a design point close to the right end of the horizontal axis). In the future, as sequential ATPG algorithms continue to improve, the complexity curve will drop lower. Also, as the scan flip-flop selection algorithms improve, for the same percentage of scan, the test generation complexity will also drop. These trends should be part of the driving forces for a wider acceptance of partial scan and sequential ATPG tools.

6. APPROACHES USING CIRCUIT MODELS AT HIGHER LEVELS OF ABSTRACTION

Test generation could be significantly sped up if a circuit model at a higher level of abstraction is used. In this section, we discuss briefly the principles of approaches using RTL models and state transitions graphs. Understanding the principles of such methods is helpful for the discussion of functional learning to be discussed later. Because the main focus of this paper is gate-level approaches, we do not intend to give a detailed survey for such approaches but only a brief description of representative methods.

Approaches using register transfer level models. Approaches using RTL models have the potential to handle larger circuits because the number of primitives in an RTL description is much smaller than the gate count. Some methods in this class of approaches use only RTL description of the circuit [Brahme and Abraham 1984; Cheng and Krishnakumar 1996; Hansen and Hayes 1995; Lee and Patel 1991; 1994; Thatte and Abraham 1980] while others assume that both gate-level and RTL models are available [Hill and Huey 1977; Breuer and Friedman 1980; Ghosh et al. 1990]. Note that automatic extraction of the RTL description from a lower level of description is still not possible and therefore the RTL descriptions must be given by the designers. It is also generally assumed that data and control are separated in the RTL model. For approaches using both RTL and gate-level models [Hill and Huey 1977; Breuer and Friedman 1980; Ghosh et al. 1990], typically a combinational test is first generated using the gate-level model. The fault-free justification sequence and the fault propagation sequence are generated using the (fault-free) RTL description. Justification and fault propagation sequences generated in such a manner may not be valid and therefore need to be verified by a fault simulator. These approaches, in general, are suitable for data-dominated circuits but not appropriate for control-dominated circuits. For approaches using only RTL models [Thatte and Abraham 1980; Brahme and Abraham 1984; Lee and Patel 1991, 1994; Cheng and Krishnakumar 1996; Hansen and Hayes 1995], functional fault models at RTL, instead of the single-stuck-at fault model at the gate-level are targeted. The approaches in Thatte and Abra-

ham [1980] and Brahme and Abraham [1984] target microprocessors, and functional fault models are defined for various functions at the control sequencing level. Tests are generated for the functional fault models and therefore a high coverage for gate-level stuck-at faults cannot be guaranteed. The methods suggested in Lee and Patel [1991, 1994] focus on minimizing the value conflicts during the value justification and fault propagation processes using the high-level information. A recently proposed technique [Hansen and Hayes 1995] can guarantee that the functional tests for their proposed functional faults achieve a complete gate-level stuck-at fault coverage. This approach also uses an efficient method for resolving the value conflicts during propagation/justification at the RT level. A method of characterizing a design's functional information using a model extended from the traditional finite state machine model with the capability of modeling both the data-path operations and the control state transitions is suggested in Cheng and Krishnakumar [1996]. However, this method does not target any fault model and only generates functional vectors for design verification.

Approaches using state transition graphs. For finite state machines (FSMs) for which the state transition graphs are available, test sequences can be derived using the state transition information. In general, this class of approaches can handle only relatively small circuits due to the known state-explosion problem in representing a sequential circuit using its state table. However, successful applications of such approaches to protocol performance testing [Sabnani and Dahbura 1988] and to testing the boundary-scan Test Access Port (TAP) controller have been reported [Dahbura et al. 1989]. The earliest method is the checking experiment [Hennie 1964] which is based on distinguishing sequences. The distinguishing sequence is defined as an input sequence that produces different output responses for each initial state of the FSM. This approach is concerned with the problem of determining whether or not a given state machine is distinguishable from all other possible machines with the same or fewer number of states. No explicit fault model is used. The distinguishing sequence may not exist and the bound on length, if it exists, is proportional to the factorial of the number of states. This method is impractical because of the long test sequence. Improved checking experiments, based on either the Simple I/O sequence [Hsieh 1971] or the Unique Input/Output (UIO) sequence [Sabnani and Dahbura 1988] of the FSM, significantly reduce the test length. A functional fault model in the state transition level has recently been used in a test generator FTG for FSMs [Cheng and Jou 1992]. In the single-state-transition (SST) fault model, a fault causes the destination state of a single state transition to be faulty. It has been shown [Cheng and Jou 1992] that the test sequence generated for the SST faults in the given state transition graph achieves high fault coverages for the single stuck-at faults as well as the transistor faults in its multi-level logic implementation. As an approximation, FTG uses the fault-free state transition graph to generate the fault propagation sequence. AccuraTest [Pomeranz and Reddy

1991] further improves the technique and uses both fault-free and faulty circuits' state transition graphs to generate accurate test sequences for the SST faults as well as for some multiple-state-transition faults.

7. SUMMARY AND CONCLUDING REMARKS

The presence of flip-flops and feedback loops substantially increases the complexity of the automatic test pattern generation problem. Due to the inherent intractability of the problem, it remains infeasible to automatically derive high quality tests for large, non-scan sequential designs. However, because of the great progress made during the past few years and the availability of robust commercial ATPG tools, the partial-scan design methodology that relies on such tools for test generation is becoming a reasonable alternative to the full-scan design methodology.

Like most other CAD tools, there are many engineering issues involved in building a test generator to handle large industrial designs. Industrial designs may contain tristate logic, bidirectional elements, gated clocks, I/O terminals, and so on. Proper modeling is required for such elements and the test generation process also needs some modification. Many of these issues are similar to those in the combinational ATPG problem that have been addressed by, for example, Breuer [1983], Ogihara et al. [1983], and Chakradhar et al. [1995].

Many sequential ATPG approaches try to extract some functional information from the gate-level circuit description in the pre-processing phase. Such information, if properly used for value justification and fault propagation, may save some unnecessary and/or repetitive computation during test generation. In Figure 20, we place several gate-level approaches in the horizontal axis, according to the degree of functional learning conducted in the pre-processing phase. To the very left, FTG [Cheng and Jou 1992] and AccuraTest [Pomeranz and Reddy 1991] extract the complete state transition graph. For these approaches, complete functional knowledge (such as the state transition graph) is learned and test generation is performed completely on the functional level. To the very right, STG3 [Cheng 1988] and HITEC [Niermann and Patel 1991] are mainly based on the gate-level information and no functional knowledge is pre-computed. STALLION [Ma et al. 1988] requires a (partial) state transition graph for state justification but uses a gate-level description for fault propagation. So the amount of functional knowledge learned in preprocessing is less than the amount learned by FTG and AccuraTest. STEED [Ghosh et al. 1991] computes and stores the ON and OFF sets of each state variable in preprocessing and intersects the appropriate sets during test generation for both state justification and fault propagation. The degree of learning is lower than STALLION.

According to the reported results, FTG, STALLION, and STEED run substantially faster than STG3 for some pure control circuits that have a small number of flip-flops but have a relatively complex structure (i.e., each next state line or primary output is a function of most of the present state

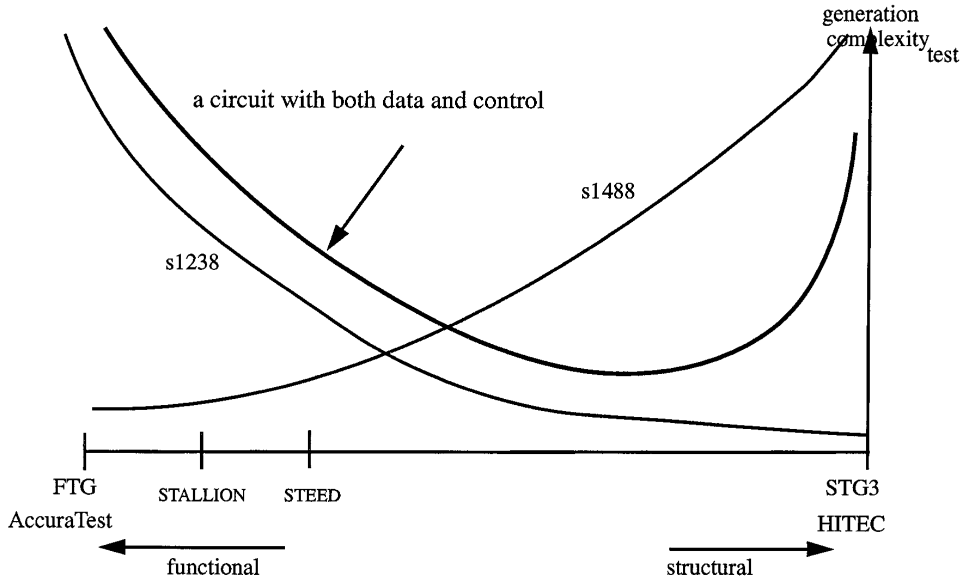

Fig. 20. Test generation approaches.

lines and the primary inputs). Such circuits include s386, s1488, and s1494 in ISCAS-89 benchmark circuits [Brglez et al. 1989]. These circuits have a small number of flip-flops and their state transition graphs are compact and easily extractable. On the other hand, STG3 and HITEC run much faster than STALLION, STEED, and FTG for circuits such as s1196, s1238, and most of the larger circuits. These circuits have a simpler circuit structure (i.e., most next state lines/primary outputs depend on a few present state lines/primary inputs), while the total number of flip-flops is relatively large and, therefore, the state transition graph is difficult to extract, store, or manipulate. It is the author's belief that for circuits having both data-path and control, the average complexity curve should resemble the one marked in Figure 20. The lowest point should be between STEED and STG3. To approach the optimal point, the test generators should be programmed to automatically identify what is the right functional knowledge to learn and to store in preprocessing, and what should be generated dynamically during test generation. To date, ATPG tools do not have such intelligence and further research is required to achieve this objective.

Developing special versions of ATPG algorithms/tools for circuits with special circuit structures and/or properties could be a good way to further improve the ATPG performance. For example, if the target circuit has a pipeline structure and is feedback-free, the algorithm described on pages 98 to 101 of Miczo [1986] is much more efficient than any algorithm surveyed in this paper that targets circuits with a more general circuit structure. Many partial-scan circuits have special circuit structures. For example, the

partial-scan circuits using the cycle-breaking strategy to select scan flip-flops do not contain cycles with more than one flip-flop. Developing algorithms and tools to target circuits with such special circuit structures is worth pursuing.

ACKNOWLEDGMENTS

The author would like to thank M. Abramovici, V. D. Agrawal, W.-T. Cheng, S. Davidson, C. J. Lin, T. Ma, F. Maamari, I. Pomeranz, and S. M. Reddy for many useful discussions on sequential ATPG during the past few years, and A. Krstic for her helpful comments and many valuable suggestions. The author would also like to thank the anonymous reviewers whose comments made the paper more balanced in the treatment of some topics.

REFERENCES

ABRAHAM, J. A. AND AGRAWAL, V. K. 1986. Test generation for digital systems. In *Fault-Tolerant Computing Theory and Techniques*, D. K. Pradhan, Ed. Prentice-Hall, Englewood Cliffs, NJ.

ABRAMOVICI, M., BREUER, M. A., AND FRIEDMAN, A. D. 1990. *Digital Systems Testing and Testable Design*. IEEE Computer Society Press, Los Alamitos, CA.

AGRAWAL, V. D. AND CHAKRADHAR, S. T. 1995. Combinational ATPG theorems for identifying untestable faults in sequential circuits. *IEEE Trans. Comput. Aided Des. 14*, 9, (Sept.), 1155–1160.

AGRAWAL, V. D. AND SETH, S. C. 1988. *Tutorial—Test Generation for VLSI Chips*. IEEE Computer Society Press, Ch. 3, Los Alamitos, CA.

BRAHME, D. AND ABRAHAM, J. A. 1984. Functional testing of microprocessors. *IEEE Trans. Comput. C-33*, 7, (July), 475–485.

BREUER, M. A. 1971. A random and an algorithmic technique for fault detection and test generation. *IEEE Trans. Comput. C-20*, (Nov.), 1366–1370.

BREUER, M. A. 1974. The effects of races, delays, and delay faults on test generation. *IEEE Trans. Comput. C-23*, 10, (Oct.), 1078–1092.

BREUER, M. A. 1983. Test generation models for busses and tri-state drivers. In *Proceedings of the 1983 IEEE ATPG Workshop* (March), 53–58.

BREUER, M. A. AND HARRISON, L. M. 1974. Procedures for eliminating static and dynamic hazards in test generation. *IEEE Trans. Comput. C-23*, 10 (Oct.), 1069–1078.

BREUER, M. A. AND FRIEDMAN, A. D. 1980. Functional level primitives in test generation. *IEEE Trans. Comput.*, (March), 223–235.

BRGLEZ, F., BRYAN, D., AND KOZMINSKI, K. 1989. Combinational profiles of sequential benchmark circuits. In *Proceedings of the 1989 IEEE International Symposium on Circuits and Systems* (May), 1929–1934.

CHAKRADHAR, S. T., ROTHWEILER, S., AND AGRAWAL, V. D. 1995. Redundancy removal and test generation for circuits with non-Boolean primitives. In *Proceedings of the 13th IEEE VLSI Test Symposium*, (April), 12–19.

CHAPPELL, S. G. 1974. LAMP: Automatic test generation for asynchronous digital circuits. *Bell Syst. Tech. J.*, (Oct.), 1477–1503.

CHENG, K.-T. 1993. Redundancy removal for sequential circuits without reset states. *IEEE Trans. CAD*, (Jan.), 13–24.

CHENG, K.-T. AND AGRAWAL, V. D. 1989. *Unified Methods for VLSI Simulation and Test Generation*. Kluwer Academic, Norwell, MA.

CHENG, K.-T. AND AGRAWAL, V. D. 1990. A partial scan method for sequential circuits with feedback. *IEEE Trans. Comput. 39*, 4, (April), 544–548.

CHENG, K.-T. AND JOU, J.-Y. 1992. A functional fault model for sequential circuits. *IEEE Trans. CAD*, (Sept.), 1065–1073.

CHENG, K.-T. AND KRISHNAKUMAR, A. S. 1996. Automatic generation of functional vectors using the extended finite state machine model. *ACM Trans. Des. Autom. Electron. Syst. 1*, 1 (Jan.), 57–79.

CHENG, K.-T. AND MA, H-K. T. 1993. On the over-specification problem in sequential ATPG algorithms. *IEEE Trans. Comput. Aided Des.* (Oct.), 1599–1604.

CHENG, W. T. 1988. The BACK algorithm for sequential test generation. In *Proceedings of the 1988 IEEE International Conference on Computer Design* (Oct.), 66–69.

CHENG, W. T. 1989. Private communications, Aug.

CHO, H., HACHTEL, G. D., SOMENZI, F. 1993. Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration. *IEEE Trans. CAD* (July), 935–945.

CHO, H., JEONG, S., SOMENZI, F., AND PIXLEY, C. 1993. Synchronizing sequences and symbolic traversal techniques in test generation. *J. Electron. Testing, Theor. Appl. 4*, 1, 19–31.

CORNO, F., PRINETTO, P., REBAUDENGO, M., REORDA, M. S., AND MOSCA, R. 1996. Advanced techniques for GA-based sequential ATPGs. In *Proceedings of the 1996 European Design and Test Conference*, (March), IEEE Computer Society Press, 375–379.

DAHBURA, A. T., UYAR, M. U., AND YAU, C. W. 1989. An optimal test sequence for the JTAG/IEEE P1149.1 test access port controller. In *Proceedings of the IEEE International Test Conference* (Aug.), 55–62.

DEVADAS, S., MA, H.-K. T., AND NEWTON, A. R. 1990. Redundancies and don't-cares in sequential logic synthesis. *J. Electron. Testing*, (Feb.), 15–30.

FUJIWARA, H. AND SHIMONO, T. 1983. On the acceleration of test generation algorithms. *IEEE Trans. Comput. C32*, (Dec.), 1137–1144.

GHOSH, A., DEVADAS, S., AND NEWTON, A. R. 1990. Sequential test generation at the register-transfer and logic levels. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, (June), 580–586.

GHOSH, A., DEVADAS, S., AND NEWTON, A. R. 1991. Test generation and verification for highly sequential circuits. *IEEE Trans. Comput. Aided Des.* (May), 652–667.

GLAESER, U. AND VIERHAUS, H. T. 1995. FOGBUSTER: An efficient algorithm for sequential test generation. In *Proceedings of the European Design Automation Conference (EURO-DAC'95)*, (Sept.), IEEE Computer Society Press, 230–235.

GOEL, P. 1981. An implicit enumeration algorithm to generate tests for combinational circuits. *IEEE Trans. Comput. C-30*, (March), 215–222.

GOLDBERG, D. E. 1989. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

GOLDSTEIN, L. H. 1979. Controllability/observability analysis for digital circuits. *IEEE Trans. Circuits Syst. CAS-26.*, (Sept.), 685–693.

GUPTA, R. AND BREUER, M. A. 1990. The ballast methodology for structured partial scan design. *IEEE Trans. Comput. 39*, 4, (April), 538–544.

HANSEN, M. C. AND HAYES, J. P. 1995. High-level test generation using symbolic scheduling. In *Proceedings of the IEEE International Test Conference*, (Oct.), 586–595.

HENNIE, F. C. 1964. Fault-detecting experiments for sequential circuits. In *Proceedings of the 5th Annual Symposium on Switching Circuit Theory and Logical Design*, (Nov.), 95–110.

HILL, F. J. AND HUEY, B. 1977. SCIRTSS: A search system for sequential circuit test sequences. *IEEE Trans. Comput. C-26*, (May), 490–502.

HSIAO, M. S., RUDNICK, E. M., AND PATEL, J. H. 1996. Alternating strategy for sequential circuit ATPG. In *Proceedings of the European Design and Test Conference* (March), IEEE Computer Society Press, 368–374.

HSIAO, M. S., RUDNICK, E. M., AND PATEL, J. H. 1996. Automatic test generation using genetically-engineered distinguishing sequences. In *Proceedings of the 14th IEEE VLSI Test Symposium*, (April), 216–223.

HSIEH, E. P. 1971. Checking experiments for sequential machines. *IEEE Trans. Comput. C-20*, (Oct.), 1152–1166.

IYER, M. A. AND ABRAMOVICI, M. 1994. Sequential untestable faults identified without search. In *Proceedings of the IEEE International Test Conference*, (Oct.), IEEE Computer Society Press, 259–266.

KELSEY, T. P., SALUJA, K. K., AND LEE, S. Y. 1993. An efficient algorithm for sequential circuit test generation. *IEEE Trans. Comput. 42*, 11, (Nov.), 1361–1371.

KEIM, M., BECKER, B., AND STENNER, B. 1996. On the (non-)resettability of synchronous sequential circuits. In *Proceedings of the IEEE VLSI Test Symposium*, (April), 240–245.

KIRKLAND, T. AND MERCER, M. R. 1987. A topological search algorithm for ATPG. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, (June), 502–508.

KUBO, H. 1968. A procedure for generating test sequences to detect sequential circuit failures. *NEC Res. & Dev.*, (Oct. 1968), 69–78.

LEE, D. H. AND REDDY, S. M. 1991. A new test generation method for sequential circuits. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD-91)*, (Nov.), 446–449.

LEE, J. AND PATEL, J. H. 1991. A signal-driven discrete relaxation technique for architectural level test generation. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, (Nov.), 458–461.

LEE, J. AND PATEL, J. H. 1994. Architectural level test generation for microprocessors. *IEEE Trans. CAD, 13*, 10 (Oct.), 1288–1300.

MA, H-K. T., DEVADAS, S., NEWTON, A. R., AND SANGIOVANNI-VINCENTELLI, A. 1988. Test generation for sequential circuit. *IEEE Trans. Comput. Aided Des.*, (Oct.), 1081–1093.

MARCHOK, T. E., EL-MALEH, A., MALY, W., AND RAJSKI, J. 1995. Complexity of sequential ATPG. In *Proceedings of the European Design and Test Conference*, (March), IEEE Computer Society Press, 252–261.

MARLETT, R. 1978. EBT, a comprehensive test generation technique for highly sequential circuits. In *Proceedings of the 15th Design Automation Conference*, (June), 332–339.

MICZO, A. 1986. *Digital Logic Testing and Simulation*, Harper and Row, New York.

MUTH, P. 1976. A nine-valued circuit model for test generation. *IEEE Trans. Comput. C-25*, (June), 630–636.

NIERMANN, T. M., CHENG, W.-T., AND PATEL, J. H. 1992. PROOFS: A fast, memory-efficient sequential circuit fault simulator. *IEEE Trans. CAD 11*, 2, (Feb.), 198–207.

NIERMANN, T. AND PATEL, J. H. 1991. HITEC: A test generation package for sequential circuits. In *Proceedings of the European Design Automation Conference*, (Feb.), 214–218.

NITTA, S., KAWAMURA, M., AND HIRABAYASHI, K. 1985. Test generation by activation and defect-drive (TEGAD). *Integration 3*, 12 (Dec.).

OGIHARA, T., MURAI, S., TAKAMATSU, Y., KINOSHITA, K., AND FUJIWARA, H. 1983. Test generation for scan design circuits with tri-state modules and bidirectional terminals. In *Proceedings of the 20th ACM/IEEE Design Automation Conference*, (June), 71–78.

PIXLEY, C. 1990. A computational theory and implementation of sequential hardware equivalence. In DIMACS Tech. Rep. 90-31. In *Workshop on Computer-Aided Verification*, vol. 2. AMS, Providence, R.I. 293–320.

PIXLEY, C. AND BEIHL, G. 1991. Calculating resettability and reset sequences. In *Proceedings of the IEEE International Conference on Aided Design*, (Nov.), 376–379.

PIXLEY, C., JEONG, S., AND HACHTEL, G. 1994. Exact calculation of synchronous sequences based on binary decision diagrams. *IEEE Trans. CAD, 13*, 8, (Aug.), 1024–1034.

POMERANZ, I. AND REDDY, S. M. 1991. On achieving a complete fault coverage for sequential machines using the transition fault model. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, (June), 341–346.

POMERANZ, I. AND REDDY, S. M. 1992. The multiple observation time test strategy. *IEEE Trans. Comput. 41*, 5, (May), 627–637.

POMERANZ, I. AND REDDY, S. M. 1993. Classification of faults in sequential circuits. *IEEE Trans. Comput. 42*, 9, (Sept.), 1066–1077.

POMERANZ, I. AND REDDY, S. M. 1994. On identifying untestable and redundant faults in synchronous sequential circuits. In *Proceedings of the 12th IEEE VLSI Test Symposium*, (April), 8–14.

POMERANZ, I. AND REDDY, S. M. 1995. LOCSTEP: A logic simulation based test generation procedure. In *Proceedings of the IEEE Fault Tolerant Computing Symposium*, (June).

PRINETTO, P., REBAUDENGO, M., AND REORDA, M. SONZA 1994. An automatic test pattern generator for large sequential circuits based on genetic algorithm. In *Proceedings of the IEEE International Test Conference*, (Oct.), 240–249.

PUTZOLU, G. R. AND ROTH, J. P. 1971. A heuristic algorithm for the testing of asynchronous circuits. *IEEE Trans. Comput. C-20*, (June), 639–647.

RHO, J., SOMENZI, F., AND PIXLEY, C. 1993. Minimum length synchronizing sequences of finite state machines. In *Proceedings of the ACM/IEEE Design Automation Conference*, (June), 463–468.

ROTH, J. P. 1966. Diagnosis of automata failures: A calculus and a method. *IBM J. Res. Dev. 10*, (July), 278–291.

RUDNICK, E. M., HOLM, J. G., SAAB, D. G., AND PATEL, J. H. 1994. Application of simple genetic algorithms to sequential circuit test generation. In *Proceedings of the European Design and Test Conference*, (March), 40–45.

RUDNICK, E. M. AND PATEL, J. H. 1995. Combining deterministic and genetic approaches for sequential circuit test generation. In *Proceedings of the ACM/IEEE 32nd Design Automation Conference*, (June), 183–188.

RUDNICK, E. M., PATEL, J. H., GREENSTEIN, G. S., AND NIERMANN, T. M. 1994. Sequential circuit test generation in a genetic algorithm framework. In *Proceedings of the ACM/IEEE Design Automation Conference*, (June), 698–704.

SAAB, D. G., SAAB, Y. G., AND ABRAHAM, J. A. 1992. CRIS: A test cultivation program for sequential VLSI circuits. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, (Nov.), 216–219.

SAAB, D. G., SAAB, Y. G., AND ABRAHAM, J. A. 1994. Iterative [simulation-based genetics + deterministic techniques] = complete ATPG. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, (Nov.), 40–43.

SABNANI, K. AND DAHBURA, A. 1988. A protocol test generation procedure. *Comput. Netw. 15*, (April), 285–297.

SCHNURMANN, H. D., LINDBLOOM, E., AND CARPENTER, R. G. 1975. The weigthed random test-pattern generator. *IEEE Trans. Comput. C-24*, 695–700.

SCHULZ, M. H. AND AUTH, E. 1988. Advanced automatic test pattern generation and redundancy identification techniques. In *Proceedings of the 18th IEEE International Symposium on Fault-Tolerant Computing*, (June), 30–35.

SCHULZ, M. H. AND AUTH, E. 1989. ESSENTIAL: An efficient self-learning test pattern generation algorithm for sequential circuits. In *Proceedings of the International Test Conference*, (Aug.), 28–37.

SCHULZ, M. H., TRISCHLER, E., AND SARFERT, T. M. 1988. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Trans. CAD*, (Jan.), 126–137.

SESHU, S. 1965. On an improved diagnosis program. *IEEE Trans. Electron. Comput. EC-14*, 2, (Feb.), 76–79.

SESHU, S. AND FREEMAN, D. N. 1962. The diagnosis of asynchronous sequential switching systems. *IRE Trans. Electron. Comput. EC-11*, (Aug.), 459–465.

SNETHEN, T. J. 1977. Simulator oriented fault test generator. In *Proceedings of the 14th ACM/IEEE Design Automation Conference*, (June), 88–93.

SRINIVAS, M. AND PATNAIK, L. M. 1993. A simulation-based test generation scheme using genetic algorithms. In *Proceedings of the IEEE International Conference VLSI Design*, (Jan.), IEEE Computer Science Press, 132–135.

THATTE, S. M. AND ABRAHAM, J. A. 1980. Test generation for microprocessors. *IEEE Trans. Comput., C-29*, 6, (June), 429–441.

THOMAS, J. J. 1971. Automated diagnostic test program for digital networks. *Comput. Des.*, (Aug.), 63–67.

WEHBEH, J. A. AND SAAB, D. G. 1994. On the initialization of sequential circuits. In *Proceedings of the IEEE International Test Conference*, (Oct.), 233–239.

Received January 1996; revised September 1996; accepted September 1996