1992

A Switch-Level Test Generation System

Kent L. Einspahr
Concordia College, eins@seward.ccsn.edu

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Proceedings, The Fifth International Conference on VLSI Design, 1992.

A Switch-Level Test Generation System

Kent L. Einspahr and Sharad C. Seth
Dept. of Computer Science and Engineering
University of Nebraska
Lincoln, NE 68588-0115

Abstract

This paper presents a switch-level test generation system for synchronous sequential circuits in which a new algorithm for switch-level test generation and an existing fault simulator are integrated. For test generation, a switch-level circuit is modeled as a logic network that correctly models all aspects of switch-level behavior. The time-frame based algorithm uses asynchronous processing within each clock phase to achieve stability in the circuit, and synchronous processing between clock phases to model the passage of time. Unlike earlier time-frame based test generators for general sequential circuits, the test generator presented uses the monotonicity of the logic network to speed up the search for a solution. Results on benchmark circuits show that the test generator outperforms an existing switch-level test generator both in time and space requirements.

Introduction

This paper summarizes an algorithm and the implementation of a Switch-LeVEL TEst generation system (SVELTE) for synchronous sequential circuits. Features of the algorithm include the ability to handle sequential as well as combinational circuits; the ability to handle both line stuck-at faults and transistor stuck-open faults; fully automatic processing; reverse time processing to generate the final test pattern first, followed by the generation of any initial test patterns that are necessary; and a combination of search-based techniques with some symbolic processing.

Earlier work in switch-level test generation dealt with only combinational functions but recognized that under stuck-open faults the circuit could exhibit a memory state. Jain and Agrawal [10] proposed a gate level model of a CMOS circuit. It allowed the use of a standard gate-level test generation algorithm and an initialization algorithm for stuck-open faults.

Another class of algorithms dealt directly with the switch level. Chen, et al. [7] developed a switch-level algorithm that generated tests for transistor stuck-open faults based upon the PODEM algorithm [9]. Their algorithm handled precharged logic but did not apply to sequential behavior. Agrawal and Reddy [1] and Reddy, et al. [13] proposed a similar algorithm based upon the D-Algorithm [15]. In addition to being limited to combinational MOS circuits, it was further limited in requiring the n-FET's to be of the same strength and the CMOS circuits to be ratio-less. These algorithms can exhibit poor performance due to the exponential growth of the number of possible paths between a module's output and power terminals. To reduce the complexity, Lioy [11] developed a mixed-level test generator which expands only the faulty block to the switch-level. The non-faulty portion of the circuit is processed at the gate level. His algorithm generates tests only for combinational circuits. Of these algorithms, only Chen, et al. report an implementation and provide performance data on small circuits.

More recently, Cho and Bryant [8] have developed a test-generation algorithm based on Bryant's switch-level simulator COSMOS [5], and have reported successful test-pattern generation for circuits with up to 770 transistors. While it improved upon previous test generators, their test generator suffers from excessive dynamic memory requirements. In addition, the amount of user intervention is also significant for large circuits. Another promising recent development is a mixed-level sequential test generator due to Chen and Abraham [6]. In its capabilities to handle a mixed gate and switch-level sequential circuit, this test generator comes closest to that described in this paper. However, it applies a nine-valued relaxation algorithm to strongly connected DC coupling components [16] whereas we use a gate-level representation of the switch-level circuit and optimize the time-frame based approach for test generation. Chen and Abraham have not integrated their test generator with a switch-level fault simulator. Reddy [14] has recently implemented a sequential test generation algorithm for MOS faults using the switch-level model of the MOSSIM-II simulator [2] and PODEM.

0-8186-2465-5/91 $1.00 © 1991 IEEE

Circuit and Fault Model

We model a circuit at two levels: a network of logic gates describing the behavior of each module locally and an interconnection of modules describing the circuit behavior globally. In general, the module-level network is an asynchronous sequential circuit capable of modeling the aspects of switch-level behavior important to fault modeling and test generation. This includes memory states (stored charge) under stuck-open faults, bidirectionality of switches, differential node and transistor strengths, etc. Between modules the signals are assumed to flow in one direction and each cycle of modules is assumed to be broken by a clocked memory element. It is also assumed that sufficient time is allowed between clock pulses for the network as a whole to stabilize as an asynchronous circuit. If the switch-level behavior is not of interest for fault modeling for specific modules, the modules could be described simply in a functional way, e.g. by gate networks guaranteed only to capture their input/output behavior.

A switch-level circuit with four modules is shown in Figure 1. The circuit accumulates the parity of A and stores it in the flip-flop to be compared with the next input on A. Modules 1-3 are channel-connected subnetworks in the circuit. Module 4 is a functional block that can be represented at a higher level than the switch-level.

A gate-level logic network describing the switch-level circuit of modules 1-3 is shown in Figure 2. While any equivalent logic network may be used to model the switch-level, we have implemented SVELTE using the description generated by the COSMOS simulator. The Boolean formulas describing each channel-connected subnetwork are expressed in the form of a directed acyclic graph (DAG). The symbols $\wedge$ and $\vee$ represent the AND and OR operations. The numbers on the internal DAG nodes in Figure 2 are used only for referencing the nodes. Two-rail logic is used to encode the 0, 1, and X states employing the following encoding scheme:

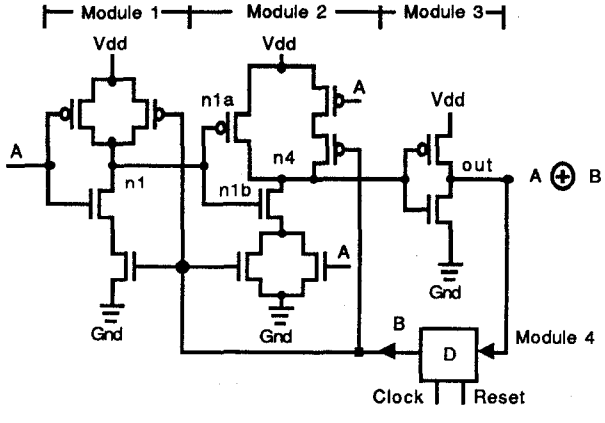

Figure 1. Parity Check Circuit. (Adapted from [17]).

A	A.0	A.1
0	1	0
1	0	1
X	1	1

Primary inputs of the circuit (e.g. node A in Figure 2) appear at leaf nodes in the DAG. Nodes which are capable of storing charge (hereafter referred to as storage nodes) also appear in the logic network. Storage nodes retain their previous value if all adjacent transistors are kept open due to the value on the transistor gate or to a fault on the transistor. Module outputs of the circuit (e.g. n1, n4 and out) are storage nodes and occur as root nodes in the DAG. The module outputs may also be connected back to the same module via 'pseudo' inputs or may be connected to the inputs of other modules. Like primary inputs, these inputs also appear at leaf nodes in the DAG.

The Boolean formula for a node in the circuit is represented by a node in the DAG and all its descendents. For example, module 1 of the circuit models a CMOS NAND structure. The corresponding DAG in Figure 2 represents the Boolean formula for the NAND. n1 = 0 (i.e. n1.0 = 1 $\wedge$ n1.1 = 0) only when both A and B are 1 as follows:

(A = 1) $\wedge$ (B = 1) =>
(A.0 = 0 $\wedge$ A.1 = 1) $\wedge$ (B.0 = 0 $\wedge$ B.1 = 1) =>
(A.1 = 1 $\wedge$ B.1 = 1) $\wedge$ (A.0 = 0 $\wedge$ B.0 = 0).

The steady state response of the circuit requires repeated evaluations of the excitation function of the circuit until it stabilizes. The logic network gives the functional representation for a single evaluation of the excitation function. The excitation function is simulated by repeatedly computing the new states for the module output nodes as a function of the logic operations and the current values of the module input nodes until a stable state is reached. Stability is checked by re-applying the values at the module inputs and applying the current values of module outputs, such as node n4 in module 2, to their

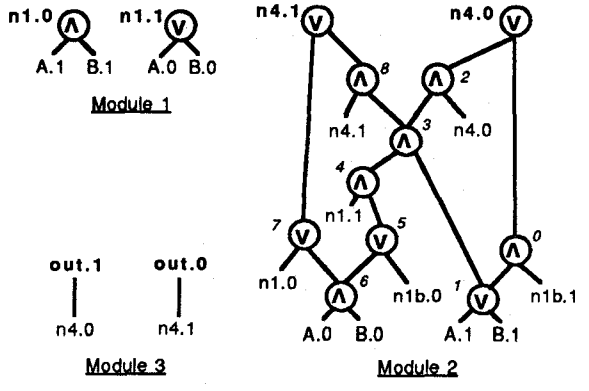

Figure 2. DAGs for parity check circuit.

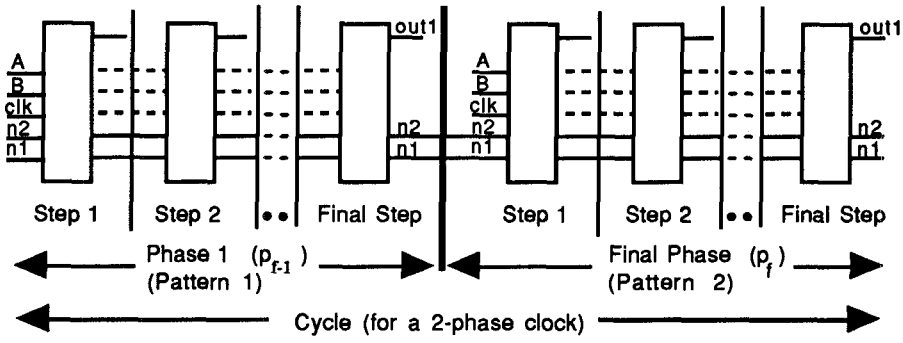

Figure 3. Time frame expansion.

'pseudo' inputs within the same module.

A line stuck-type fault on a primary input of the circuit or on a storage node output is modeled by faulting each occurrence of the node across all modules. A similar fault on a fanout of a primary input or storage node is modeled by faulting only the leaf node of the module for which the faulty fanout is an input. Stuck-open faults are modeled as line stuck-at faults. For example, a stuck-at-0 fault on the fanout line leading to gate n1b and a stuck-open fault on the transistor controlled by that same gate are both modeled as a fault on leaf nodes n1b.0 and n1b.1 in the DAG, requiring faulty value assignments of 1 and 0 to those leaf nodes.

Algorithm

A method of *reverse time processing* [12] is used by SVELTE to generate the test patterns. The time frame expansion used by SVELTE is illustrated in Figure 3 for a hypothetical circuit with primary inputs A, B, and clk; storage nodes n1 and n2; and primary output out1. We will define the terms *cycle*, *phase*, and *step* as follows:

cycle	A complete sequencing of the clocks.
phase	A period in which the circuit reaches stability after clock values (and possibly input values) changed.
step	A single evaluation of the DAG nodes.

In the reverse time processing of SVELTE, test generation begins in the final phase and is followed by the generation of any initial test patterns that may be necessary. During any one of the phases in which a test pattern is generated, processing begins at an arbitrary step s of the phase. Using implication and justification, a set of assignments to the inputs and storage nodes of all modules in step s is obtained. The set of assignments is implicated to subsequent steps (s+1, .. , s_f) in the phase until stability is determined by two successive steps having the same state. To ensure stability independent of race conditions, the circuit must remain stable during the transition states that occur when the clock and input values change between phases. The transition between two successive phases is handled in the final step of the first of the two phases.

If a storage node needs to be initialized upon entry to a phase, a previous test pattern (and thus, a previous phase) is necessary to initialize the specified nodes. Appropriate justification is performed to determine a satisfying set of assignments. When any inconsistency is reached, backtracking occurs within the phase to adjust the current assignments and search path. If no assignment is possible in the current phase, backtracking may proceed across the phase boundary to modify the assignments in a previously processed phase.

The SVELTE test generation algorithm, shown in Figure 4, initially selects an output of the faulty module to which the fault effect will be propagated. As long as the logic model contains only AND and OR operations, and inverted polarities of line values between modules are handled by the dual-rail logic or by the test generator, the logic model is monotonic and the values of each output node bit can immediately be assigned the value of a target-fault bit of which it is a function. For example, if a stuck-open fault on n1b is tested (Fig. 1), SVELTE sets the good/faulty values of n1b.0 and n1b.1 to 1/0 and 0/1, respectively. The values of n4.0 and n4.1 can immediately be assigned 1/0 and 0/1. The assignment to the n4 output allows processing to take place not only from the inputs toward the outputs but also from the outputs toward the inputs, giving the algorithm a 'two-directional' advantage over other search-based techniques such as PODEM or the D-Algorithm in which the frontiers advance primarily in only one direction.

In the two-directional processing, the target-fault bit values are implicated to ancestor nodes by applying the function of each parent to the current values of its children. At the same time, deterministic justification proceeds from the selected output to determine the necessary values of one or all children based upon the value of the node, its logic operation, and the values of its children. If no test is found by propagating the fault to the selected output, another output is selected. If all outputs

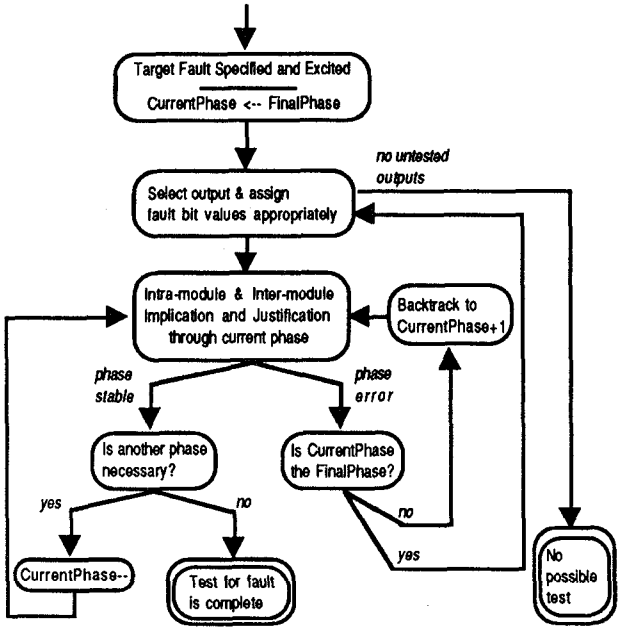

Figure 4. SVELTE Test Generation Algorithm.

have been selected and no test is found, the algorithm attempts to propagate the fault value in the final phase from a selected storage node rather than from the target fault, requiring a preliminary test pattern in which the fault effect is propagated to that selected storage node. If a sequence of test patterns is still not found, SVELTE returns a 'no possible test' response.

In the basic loop of the algorithm, justification and implication begin in those modules that have faulty values assigned or have outputs assigned due to initial faulty values. Justification and implication continue within each individual module until the module outputs are fully justified, or until deterministic processing can no longer proceed after which non-deterministic justification may begin. When values are assigned to primary inputs, they are immediately implicated to the other modules in the circuit. On the other hand, storage node values are not propagated to other modules until the respective module is justified and stabilized. Once a module has been justified and checked for stability the module inputs and outputs are propagated to the other modules in the circuit. The occurrence of inconsistent or bad assignments results in backtracking within the phase. In Figure 4, backtracking within the same phase is implicit in the implication/justification block. Backtracking may also proceed across phase boundaries in which case a previously determined test pattern will need to be modified. Backtracking across phase boundaries is shown in the figure beginning at the arrow marked 'phase error'. Processing in the current phase continues until all modules are stable at which point SVELTE determines whether a preliminary test pattern is required or whether the sequence of test patterns is complete. If a preliminary test is required, the algorithm moves to the previous phase and executes the basic loop to generate a pattern in that phase. If the test is complete, SVELTE returns the sequence of patterns as a test for the fault.

The functional description of the switch-level circuit allows test generation to occur at a higher level, much the same as that of gate-level models, and still correctly model the switch-level behavior. Bryant has shown that the size of a functional description for most switch-level circuits with n transistors is O(n) [3,4].

Throughout the program, the strengths of search-based techniques are used to improve performance. Since SVELTE processes each module separately and stabilizes each module before propagating its values to the other modules in the circuit, it is able to stop processing one module when a non-deterministic decision must be made and continue processing a module whose next assignment may be deterministic. Often, delaying a non-deterministic choice in a module A and doing the deterministic processing in another module results in an assignment that resolves the non-determinism in module A. Delayed processing can also take place across phase boundaries, as will be shown later in the example, by treating the module inputs as variables. Treating the nodes in this way is different than typical search-based methods which would normally make a guess for the assignment. SVELTE essentially handles the formula symbolically, allowing several node assignments to be tested at the same time. That is, if node A is delayed and is treated as a variable, then subsequent tests actually test the cases when A=1 and A=0. As a result of the delayed decision, a previous phase may determine the value of the variables in question during the current phase.

The search-based approach also allows selective processing based upon factors such as distance from a specific node, or upon the number of children or parents of a node. Distance measurement heuristics have been implemented to select the order of the fault propagation paths that are tested. The search involved in testing undetectable faults has been reduced by maintaining information about states which lead to no possible test and by employing two-directional processing in which a preliminary step is simulated in the first phase prior to performing reverse time processing.

SVELTE seeks to find a single test for a fault based upon the information contained in the logic model itself. It is fully automatic and requires no user intervention.

Example

For this example the clock cycle will be specified by 10 where 1 and 0 are the values of the clock in the first and final phases of the clock, respectively. Assume the delay module is a master-slave flip-flop.

Circ	# tra	# mod	fault total	fault detect	time (sec) SVELTE	time (sec) COSMOS	# test patterns SVELTE	# test patterns COSMOS
mc8	149	34	198	190	24.7	25.0	34	15
mc14	257	58	342	328	83.9	300.3	58	24
mc16	293	64	390	374	113.1	aborted	66	aborted
mc32	581	130	774	742	649.8	aborted	130	aborted
mc48	869	194	1158	1110	1922.1	aborted	194	aborted
sr4	72	9	72	34	24.8	46.1	8	8
sr8	144	18	136	66	96.0	169.0	16	16
sr16	288	36	264	130	465.7	718.0	32	32
srcadd4	184	25	190	152	208.1	212.0	40	24

Table 1. Statistics and comparative results.

Let transistor n1b be stuck-open in Figure 1. Due to the monotonicity of the DAG, SVELTE is able to immediately assign the n1b.1 (n1b.0) good/faulty value of 1/0 (0/1) to the module 2 output, n4.0 (n4.1), in the final phase p_f since the fault effect must be propagated through the n4 module output to reach the primary output node. Leaf node n1 in module 2 has the same good circuit value as n1b (i.e. n1.0 = 0, n1.1 = 1). The fault value at n1b.0 is propagated through nodes 5,4,3, and 8 to the root node n4.1. The fault value at n1b.1 is propagated through node 0 to n4.0. Justification from the root nodes results in assignments of X/0 and X/1 to the n4.0 and n4.1 leaf values in module 2, indicating that n4 must be initialized to 1 in the faulty circuit by a preliminary test pattern. Deterministic processing is completed in this phase after propagation of the n1 values from module 1 to module 2. At this point, a non-deterministic decision exists for the assignments of A and B; one node must be 0 and the other must be 1.

Initialization of the faulty value for node n4 occurs in phase p_{f-1}. SVELTE always tries to assign values to primary inputs when possible rather than to storage nodes. As a result, the faulty value of 1 (0) at the n4.1 (n4.0) output in this phase will come from the children of node 6 (1) so that no other storage nodes need to be assigned. Thus, in the faulty circuit, A.1 = B.1 = 0 and A.0 = B.0 = 1 so by dual-rail logic, A = B = 0. The value of out in the faulty circuit will also be 0 causing B to take on that value in phase p_f. The ability to delay the non-deterministic decision regarding A and B in phase p_f has resulted in the deterministic assignment of B=0 and A=1 (i.e. B.0=1, B.1=0, A.0=0, and A.1=1). Since B must be assigned a value 0 in phase p_{f-1} the reset input of the delay flip-flop must be set to 1.

In summary, the test patterns found by SVELTE for the n1b stuck-open fault are:

Phase	Input <A,B,clk,reset,n1,n4>	Output <n1,n4,out>
p_{f-1}	<0,0,1,1,X,X>	<1,1,0>
p_f	<1,0,0,0,1,1>	<1,0/1,1/0>

Implementation and Results

In the implementation of the SVELTE system, the test generator is an independent program. It reads as input the logic network information for the circuit from which it derives its fault list. Although test patterns can be generated for each fault in the fault list, it is normally desired to use a fault simulator to determine the other faults that a test pattern detects and eliminate them from the list. We have interfaced the SVELTE test generator with the COSMOS fault simulator by executing the fault simulator as a child process.

We are currently obtaining results for a prototype implementation of the SVELTE test generator. Circuit statistics and test results for increasing sizes of a Manchester carry chain adapted from [17], for increasing sizes of a sequential shift register, and for a 4-bit sequential ripple-carry adder are shown in Table 1. All test circuits are specified completely at the switch-level. The increasing size of the test circuits provide an experimental indication of SVELTE's time complexity in comparison to Cho and Bryant's test generator (shown as COSMOS in Table 1). The results were obtained for COSMOS by using initialization variables for all input variables and using essentially no user intervention. Since SVELTE is an automatic test pattern generator, our intent was to compare SVELTE to the automatic mode of COSMOS. At the expense of user intervention the COSMOS results may be improved. SVELTE does have the ability to allow user intervention to guide test generation for hard to detect faults. For the combinational circuits tested, while SVELTE is slightly slower than COSMOS for smaller circuits, its time complexity with respect to circuit size is approximately quadratic as opposed to cubic for COSMOS. The results of the sequential circuits shown indicate that for these circuits the complexities of the test generators are the same, approximately quadratic. Very little effort has gone into optimizing SVELTE; with projected optimization we can expect the performance of SVELTE to continue to improve. These results were obtained on a SPARC workstation rated at 16 MIPs with 8Mb of memory.

SVELTE and COSMOS do generate the same fault list and find the same undetectable faults. Note that for circuits larger than the 14-bit Manchester carry chain, no results were obtained with COSMOS due to memory thrashing due to the large space requirements of that test generator. This result lends credence to the smaller memory requirements of SVELTE. COSMOS does generate a more compact set of test patterns than SVELTE as a consequence of its ability to find all the tests for a fault.

As a result of the incomplete state of the SVELTE mixed-level test generation mode, we have not been able to obtain a good comparison with the results of the Chen and Abraham test generator.

Conclusion

SVELTE is an efficient test generator. All line stuck-at and transistor stuck-open faults are handled in clocked sequential circuits. Use of a logic network description of the switch-level circuit ensures that all other aspects of switch-level network behavior are handled. With a well-selected model, the size of the logic network used by SVELTE is O(n) for most circuits with n transistors. The algorithm exploits the functional description of the switch-level circuit. SVELTE is a fully automatic test pattern generation system that can be interfaced with a switch-level fault simulator. Preliminary results obtained with SVELTE show that its performance could be significantly better than other existing programs.

Testing of both combinational and sequential benchmark circuits continues with the current SVELTE implementation. We also plan to extend the SVELTE algorithm to a mixed-level test generator as well as incorporate several strategies to enhance non-deterministic processing.

References

[1] P. Agrawal, S.M. Reddy, "Test Generation at MOS Level," Int'l. Conf. on Computers, Systems & Signal Processing, Bangalore, India, Dec. 10-12, 1984, pp. 1116-1119.

[2] R.E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," IEEE Trans. on Computers, Vol. C-33, No. 2, Feb. 1984, pp. 160-177.

[3] R.E. Bryant, "Algorithmic Aspects of Symbolic Switch Network Analysis," IEEE Trans. on Computer-Aided Design of Integrated Circuits, July 1987, pp. 618-633.

[4] R.E. Bryant, "Boolean Analysis of MOS Circuits," IEEE Trans. on Computer-Aided Design of Integrated Circuits, July 1987, pp. 634-649.

[5] R.E. Bryant, D. Beatty, K. Brace, K. Cho, T. Sheffler, "COSMOS: A Compiled Simulator for MOS Circuits," 24th Design Automation Conference, 1987, pp. 9-16.

[6] C. Chen, J.A. Abraham, "Mixed-Level Sequential Test Generation Using a Nine-Valued Relaxation Algorithm," International Conference on Computer-Aided Design, 1990, pp. 230-233.

[7] H.H Chen, R.G. Mathews, J.A. Newkirk, "An Algorithm to Generate Tests For MOS Circuits at the Switch Level," IEEE Int'l Test Conference, 1985, pp. 304-312.

[8] K. Cho, R.E. Bryant, "Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation", 26th Design Automation Conference, June 1989,pp. 418-423.

[9] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, Vol. C-30, No. 3, March 1981, 215-222.

[10] S.K. Jain, V.D. Agrawal, "Test Generation for MOS Circuits Using D-Algorithm," 20th Design Automation Conference, Miami Beach, FL., June 27-29, 1983, pp. 1-7.

[11] A. Lioy, "Mixed Level Test Generation For MOS Circuits", 1st European Test Conf., Paris, April 1989, pp. 208-211.

[12] R.A. Marlett, "An Effective Test Generation System for Sequential Circuits", 23rd Design Automation Conf., June 1986, pp. 250-256.

[13] M.K. Reddy, S.M. Reddy, P. Agrawal, "Transistor Level Test Generation for MOS Circuits," 22nd Design Automation Conference, 1985, pp. 825-828.

[14] M.K. Reddy, "Testable CMOS Digital Designs and Switch-Level Test Generation for MOS Digital Circuits," Ph.D. Dissertation, Univ. of Iowa, May 1991, pp. 305.

[15] J.P. Roth, "Diagnosis of automata failures: A calculus and a method", IBM J. Res. Develop., vol. 10, July 1966, pp. 278-291.

[16] A.E. Ruehli, *Circuit Analysis, Simulation and Design*, Elsevier Science Publishing Company, 1987, June 1976, pp. 630-636.

[17] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1985, pp 198,814.