TEST PATTERN GENERATION FOR VLSI : STATUS AND TRENDS

J. Rivierre

Rapporteurs: Mr. J.G. Givens
Mr. K. Heron

Abstract:

What used to be an accessible component in the MSI AND LSI periods is now embedded within a VLSI chip and surrounded by hundreds of similar components which cannot any longer be physically accessed. This is making test pattern generation one of the most challenging problems of the VLSI era.

The present methods and techniques used for test pattern generation will be recalled. Emphasis will be given to their respective strengths and limits when used for VLSI chips. Although it will be addressed, showing the need for novel approaches to cope with the VLSI challenge.

Under the new concept of design for testability, techniques such as scan path and signature testing will be developed as representing the present trends in the industry.

Introduction

An integral part of designing a network is the ability to test it such that the faulty ones can be detected and eliminated. Prior to LSI, elementary circuits or gates could be tested separately using simple methods at the lowest level of packaging while more elaborated methods were devised for large clusters of gates at higher levels of packaging. While the LSI era has forced some refinements into the methods used for cards and boards for the generation of test patterns, no major theoretical breakthroughs have occurred in the last decade.

We are now moving from LSI to VLSI and the problem of test pattern generation, not fully resolved for LSI, is becoming even more crucial. It is imperative today that the designer be aware of the concepts, the capabilities and limits, the cost of fault detection, and also of the new techniques under development, in order to produce designs testable in a cost effective way.

In the first part of this paper we will review the techniques developed and used in the industry for LSI circuits. Emphasis will be

placed on their present limits and weaknesses. Cost figures will also be given as a function of circuit count.

The second part will address novel approaches and techniques, already in use or under development to cope with the VLSI Testing challenge. These techniques such as Level Sensitive Scan Design, Scan Path, Signature analysis, which imply the need to consider testability at the early stages of a design, are part of the general concept of "Design for Testability".

Faults, Defects and Test

We will start by making a distinction between two terms that may appear equivalent : fault and defect.

A fault in a network will yield an incorrect result for a given input pattern. A defect on the other hand may or may not cause a fault. On an integrated circuit, this could be an open diode, shorted nets, a high value resistor, an underetched contact, etc.

A test for a defect is a set of conditions that would cause the defect to reveal itself as a fault if it were present. Stimulus and response are the two constitutive elements of a test. In the case of logic networks, the stimulus is a sequence of binary values to be applied to the input pins. The response is the set of binary values that appear at the output pins.

Test generation is the process of developing these stimulus - response pairs.

Exhaustive Testing

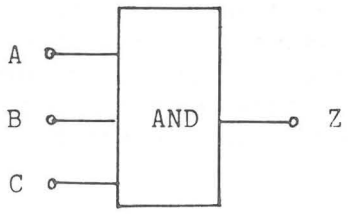
In the pre-LSI days, we tested the parts for all possible combinations. Today, for a 50-input combinational network, the number of tests would be 2^{50} or about 10^{15}. Even if a test could be applied every 10 ns, it would take 4 months to execute.

A solution to the exhaustive testing problem is to treat the logic network the way we treat assemblies. That is, test that each logic gate is good, that they are inter-connected correctly, and then by implication, the total network is good.

Fault-Model

The high number of potential defects per gate (about 30 in present technologies) necessitates to represent them by a model. The model most commonly used throughout the industry to represent defects is the Stuck-At model. The Stuck-At model assumes that a logic gate input or output is fixed to either a logic zero or a logic one.

Let us consider a single 3-input AND gate. Figure 1 shows a list of tests which detect all possible stuck faults for this gate.



INPUTS	EXPECTED RESPONSE	FAULTY RESPONSE	FAULTS DETECTED
A B C	Z	Z	
0 1 1	0	1	A - sa1, Z - sa1
1 0 1	0	1	B - sa1, Z - sa1
1 1 0	0	1	C - sa1, Z - sa1
1 1 1	1	0	Z - sa0, (A,B,C)-sa0

sa = Stuck-At

Figure 1

Let us now consider, as an example, a chip consisting of 2000 AND gates and having 50 inputs. Assuming we can somehow apply each of the 4 basic tests to each of the gates and observe the output of each gate, the number of tests will be 2000 x 4 = 8000, a very important reduction with regard to the 10^{15} required for exhaustive testing. However, the Stuck-At model does not acurately represent all possible defects which may cause faults. For example, the model does not include shorts between nets. But, historically a test with a high level of Stuck-At fault coverage has resulted in acceptable defect levels being shipped.

Nevertheless new circuit types (e.g. three states) and new technologies may be sensitive to other types of defects which cannot be modelled as stuck faults [1]. It has been recommended (Fault Tolerant VLSI Design Workshop, April 1980) that more research work be devoted to the area of fault modelling.

Another apparent weakness of the Stuck-At model is the following : when generating tests for a faulty network, the faulty network consists of the good network with only one Stuck-At fault present. This assumption is imposed since it is absolutely impractical to generate tests for networks having all the possible combinations of multiple faults. Here again history has proved that the single Stuck-At fault assumption in prior technologies is adequate. But this does not mean in any case that this adequacy would also apply for new circuit types and new technologies, and for these the implications of the single Stuck-At fault assumption should not be overlooked.

Test Generation Systems

Most test generation systems contain two sets of programs, test pattern generators and fault simulators. The test pattern generator analyzes a model of the network to produce a test set. The fault simulator evaluates the test set to determine the fault coverage and list of undetected faults. Each of these will now be described.

Test Generators

Several methods to perform automatic test generation have been proposed. They can be classified in two categories : the synthetic methods and the analytic methods.

Synthetic methods are based upon the checking of the state-tables of the networks and of the transitions from state to state. The most well known are the Hennie 2 and Poage 3 methods. These methods are well adapted to small networks but, in general, are heuristic in nature and are much harder to implement than the analytic methods.

The most commonly used analytic methods are the path sensitizing, pseudo-random, D-algorithm, manual patterns, and methods directly derived from the above. Pseudo-random pattern generation is fast and economic but has severe limitations with high fan-in circuits. The path sensitizing method implies the use of algorithms and of heuristics. But it cannot detect faults which require to sensitize several paths at a time. The D-algorithm [4] may be considered to be a formalized version of the path sensitizing method with the added intelligence necessary to sensitize multiple paths.

In a test generation system, these methods are exercized concurrently for a given network to achieve a sufficient test coverage.

In order to provide an appreciation for the limits and the capabilities of these methods and understand where further development is necessary, we will now describe the D-algorithm, on the example of figure 2.

The algorithm uses a 5-valued algebra as follows:

0	Normal meaning
1	Normal meaning
X	Unknown or don't care (0 or 1)
D	1 if no fault is present, 0 if the fault is present
$\overline{D}$	0 if no fault is present, 1 if the fault is present

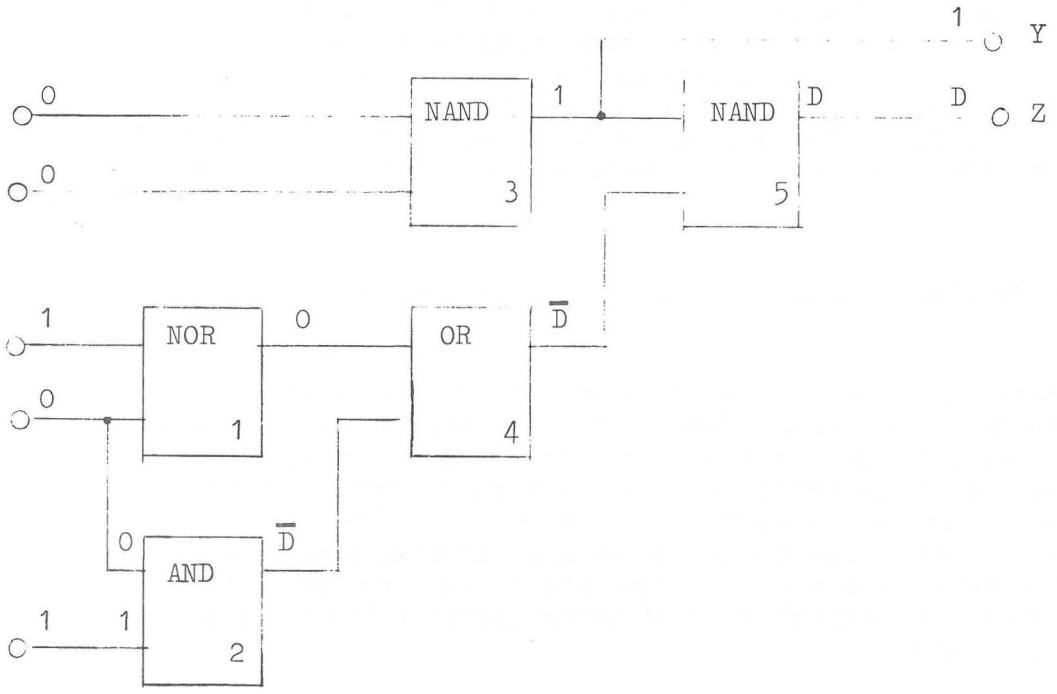


Figure 2 D-algorithm

Let us assume we want to generate a test for a Stuck-At 1 fault (D) at the output of gate 2. The process consists of the following three steps:

Step 1: apply the values at the inputs of gate 2 that will make the fault appear if it is present (A4 = 0, A5 = 1).

Step 2: drive the D or $\bar{D}$ forward from the gate under test towards a primary output, assigning sensitizing values along the way. In this case, we first go to gate 4. If the top input to gate 4 was a 1, the output would be a 1 independently of the test value on the bottom input. Therefore, the top input's sensitizing value must be 0. This causes the output of gate 4 to become D. We next trace to gate 5, and make the top input a 1, its output then becomes a D.

Step 3: justify the internal net values by driving backward towards the inputs, assigning input values to the gates which will produce the desired value. For gate 3, this implies that A1 or A2 or both be assigned a 0. We arbitrarily choose to assign both. For gate 1, the remaining non assigned input must be assigned a 1.

The test is then A1 = 0, A2 = 0, A3 = 1, A4 = 0, A5 = 1.

Unfortunately, the above algorithm is usually complicated by the fact that critical choices are frequently involved in tracing forward to a primary output, and in selecting gate input values during justification. When incorrect choices are made, the process must allow back-up and retry with other choices. The number of combinations to retry can be quite large, inducing large execution times. To minimize this problem, the algorithm implementations usually incorporate heuristics that bring global information to the major decision points.

The handling of sequential networks is considerably more difficult. Globally it implies the transformation of the network into a combinatorial one by multiple replications (time-images) of the original network.

Figure 3 shows a graph of the ability of today's automatic test generators to generate tests as a function of the number of gates for combinatorial and sequential networks.

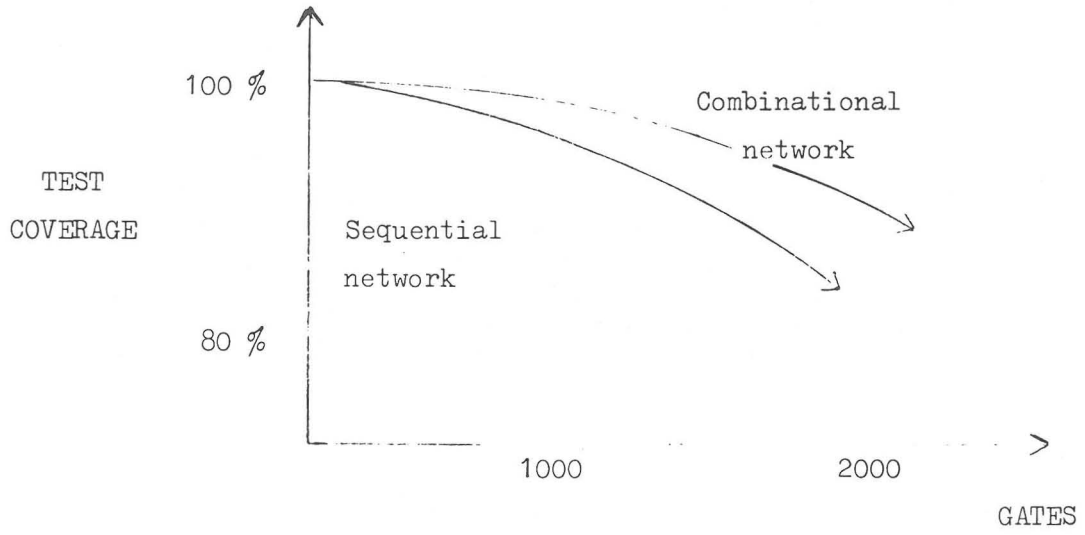


Figure 3

The graph shows that as the size of a general sequential network gets in the range of 1000 to 2000 gates, the ability to generate tests decreases to unacceptable levels. If the network is combinatorial, however, the test generation algorithms are adequate for networks of 5000 gates and above. This fact is exploited in the next part of this paper, in which the sequential networks are basically reduced to combinatorial networks.

Fault-Simulators

Fault-simulators perform the main function of determining for a given set of test patterns what faults have been undetected and of computing the test coverage.

Assuming a network with 5000 stuck faults, fault simulation is then the process of applying every given test pattern to the fault free network and to each of the 5000 copies of the good network containing one and only one of the Stuck-At faults. Parallel fault simulation uses the word size N of the computer to process N faulty networks at a time.

Deductive fault simulation [5] which allows all the faults to be simulated in one pass is faster, but is much harder to implement and requires enormous memory capacity for large circuits.

But even with the implementation of these techniques, gate level fault simulation, although deterministic in nature, is still an expensive task as we will see now.

Test Generation Cost

The test generation problem, like other classes of DA problems, has been proven to be theoretically NP-complete [6]. This implies an exponential growth in test pattern generation costs (cost here is considered equivalent to computer run time) with increasing gate count G. Of course, the use of heuristics mentioned earlier allows us to achieve a slower than exponential growth rate, and analysis of practical results [7] with the state of the art test generation methods and gate counts under 100,000 show that for the combinatorial networks:

(a) test pattern generation cost grows as G^2,

(b) parallel fault simulation cost grows as G^3 ,

(c) deductive fault simulation cost grows as G^2,

(d) test data volume grows as G.

With this G^2-growth, the change in going from 1000 to 100,000 gates will result in a 10,000 fold increase in test pattern generation and fault simulation cost. With cost of the order of minutes of CPU time for a 1000 gate structure, CPU time will reach the thousands of hours for the 100,000 gate structure !

This clearly means that test generation for combinational logic circuits is not a solved problem. This fact has been understood for some time by the companies (mainly mainframe manufacturers) which have used automatic test-pattern generation at the card and board level before. And a lot of thought has already been devoted to overcome the difficulties : "design for testability" is the generic term which covers all the techniques elaborated as potential solutions to the testing of VLSI.

Design for Testability

Test points, degating, multiplexing, control lines, ...etc., are techniques [8, 9, 10] developed for cards and boards to improve their testability and which are now being applied for VLSI. These techniques are often called "Unstructured" or "Ad-hoc" since their application to a design is specific to this design, and cannot be generalized and implemented as a test generation system.

These techniques are "after the fact", since they are generally implemented after the logic design has been done. Although they provide some help to improve the testability, and should not be neglected, they are not by themselves sufficient to solve the problem of testing VLSI and we will not elaborate more on them here.

It is widely recognized today that testability has to be taken into account at the logic design phase and that rigorous design practices [8] are necessary to insure testability, both at the generation and manufacturing levels and to allow the development of test generation systems.

These so-called "structured techniques" can be divided into three categories : scan path techniques, new concepts, and compound techniques.

Scan Path Techniques

In this category, we place the techniques that capitalize on the current test pattern generation methods which have been reviewed in the first part. They are built upon the concept that, with some additional circuitry, all the memory elements of a network can be made directly accessible. Probably the best known and most widely practiced implementation is IBM LSSD (Level Sensitive Scan Design). It consists in tieing together all the memory elements, in order to create a shift register (Fig. 4). A control signal is provided to switch the memory elements from their normal mode of operation to the shift register mode. With a shift input and a shift output accessible at the circuit pins, the states of the memory elements can be set and read directly by shifting data in and out.

Moreover, the possibility of implementing several shift register chains, which results in partitioning the combinatorial circuits into smaller independent clusters will lower the effects of the G^2 curve in terms of cost.

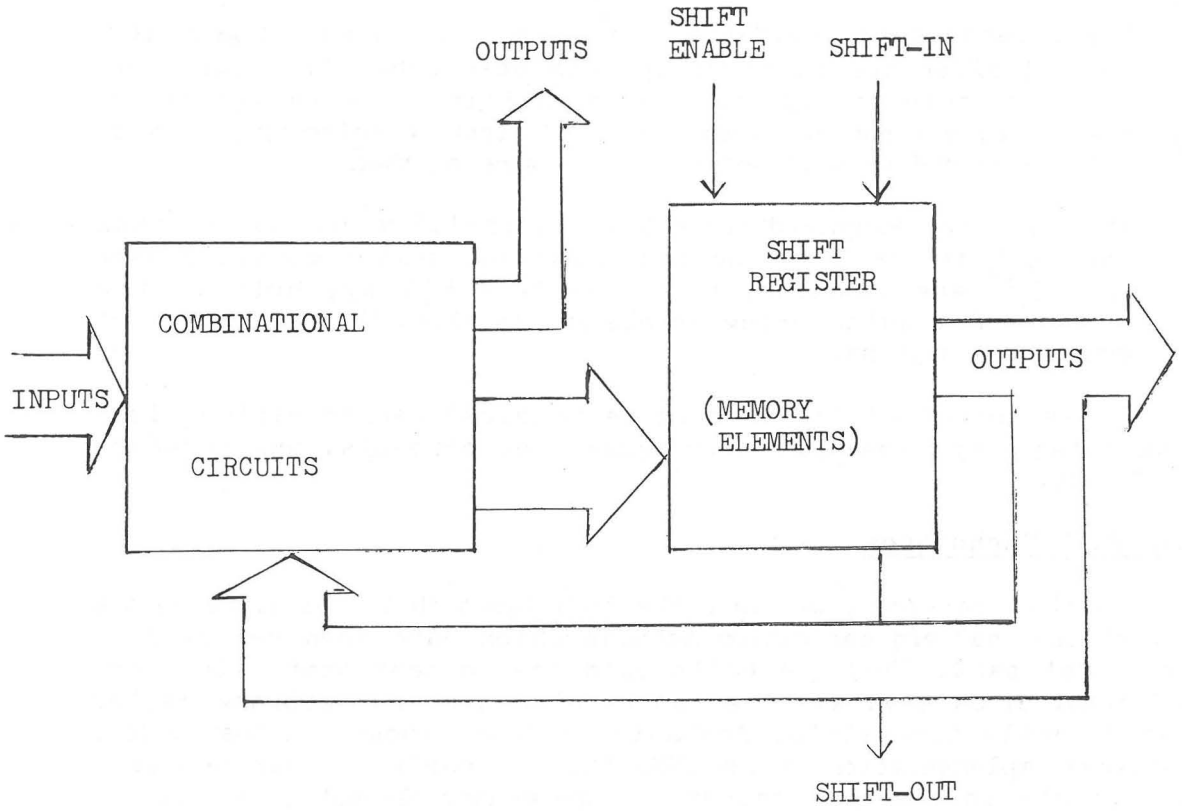


Figure 4

Another example of a method to achieve direct accessibility of the memory elements, is called Random Access Scan [8, 11]. It consists of implementing all the memory elements in a RAM, and by adding X and Y decoders on-chip. With data-in and data-out signals accessible at the circuit pins, all the memory elements can be written and read individually.

The net effect of these implementations is that the network can now be considered as combinational, the memory elements playing the role of extra virtual input and output pins. The "classical" techniques of test pattern generation and fault simulation can then be used again.

New Concepts

The second category includes techniques which do not capitalize on past developments, but approach testability with new principles.

The most appealing technique which has emerged for VLSI in the late 70's is the Built-in Test, also called Self-test. It consists in implementating hardware test pattern generators and test data checkers directly on the integrated circuit. (Fig. 5).

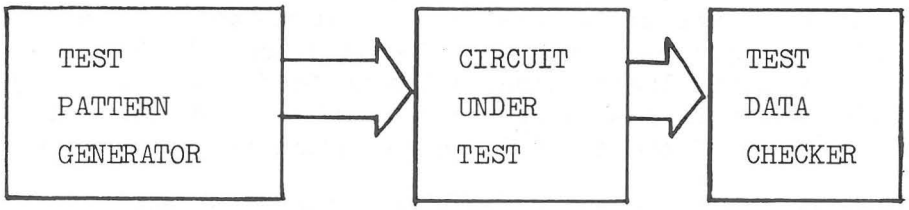


Figure 5

In order for this technique to be practical, the extra hardware required has to be kept within reasonable limits.

For the test data checker, this can be achieved by the use of data compression techniques. An example implementation, called Signature Testing, uses linear feedback shift registers. From the theory of linear machines [12], the linear shift register of figure 6, where Z1 to Z8 are the outputs of the circuit under test, acts as a polynomial divider of the output sequence and will store the residue of this division, called the signature of the test.

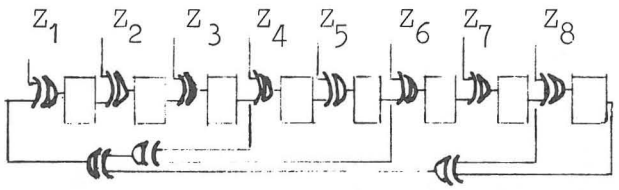


Figure 6

It can be shown that the probability that a faulty circuit is not detected can be as low as 2^{-m}, where m is the length of the linear feedback shift register. This is of course dependent upon the choice of the polynomial divider (it has to be a primitive polynomial) and of the completeness of the input sequence.

The on-chip generation of test sequence can be realized by using parts of the active logic (i.e. ROM's), but this is design dependent and does not lend itself to systematic implementation. Linear feedback shift registers can again be used here. With its parallel inputs tied to a given logic value, a linear feedback shift register can be made to develop a sequence of patterns of maximal length $2^m - 1$, at the rate of the clock, with the adequate feedback lines.

Besides the extra hardware required, the main drawbacks, today, of the built-in test described here are due:

(a) to the considerable data compression which does not allow for any diagnostics and does not provide any fault coverage figure;

(b) to the pseudo random test sequences which are known to have some weaknesses to detect faults for circuits with high fan-in.

Compound Techniques

The need to overcome the above drawbacks has led researchers to come up with an approach compounded of scan path and linear feedback shift registers. It is called BILBO : Built-in Logic Block Observation [8, 13].

Each register (Fig. 7) is composed of a string of latches with some extra logic for shift and feedback, and two extra inputs B1 and B2 to control the 4 modes of operation of this register:

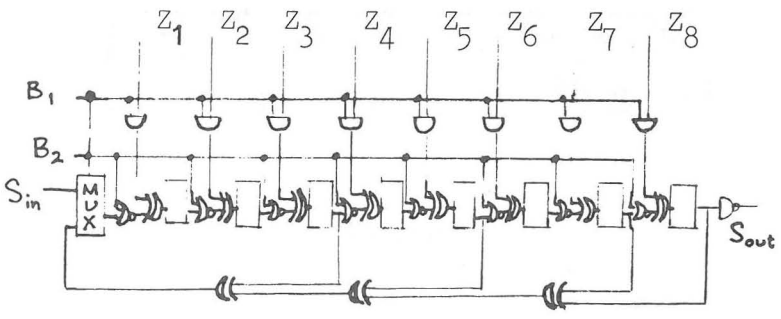


Figure 7

(i) B1, B2 = 1,1. This is the system operation mode. All the latches take the data from the inputs Z, and after the clock, these data will appear on the output Q.

(ii) B1, B2 = 0,0. The register acts then as LSSD shift register.

(iii) B1, B2 = 1,0. The register is then a linear feedback shift register which can be used as a signature register or as a test pattern generator by setting its inputs to a given logic value.

(iv) The 4th mode (0,1) will reset all the latches.

The major merit of this technique may very well be that it provides a bridge between techniques which have already been implemented into test generation systems and the new techniques. This bridge is likely to provide an easier and smoother implementation of these new techniques into automatic systems and broaden their usage much faster.

Concluding Remarks

The intent of this paper, besides a quick review of the evolution and the state of the art of test generation techniques, was to point out the areas where more research work is felt necessary, such as fault modelling, implementation of test generation algorithms, development of new techniques and integration of these into fully automated test generation systems.

Also, aside of the research work, it is felt that students specializing in integrated circuits should be educated in test generation to a level at which they will appreciate the need to integrate the solution of testing problems at the logic design stage and be able to choose and implement the most appropriate technique.

Acknowledgements

I would like to express all gratitude to Thomas Williams, from IBM Boulder, who made available to me the results of his thorough study and analysis of test generation throughout the industry. The results of his findings have been widely used in the writing of this note.

References

[1] Elzig, Y., "Automatic Generation for Stuck-open faults in CMOS VLSI", 18th DA Conference - June 1981.

[2] Hennie, F.C., "Finite State Models for Logical Machines", J. Wiley & Sons, N.Y., 1968.

[3] Poage, J.F. and E.J. McCluskey, "Derivation of Optimum tests for Sequential Machines", Proc. Fifth Annual Symp. on Switching Circuit Theory and Logic Design, 1964.

[4] Roth, J.P., "Diagnosis of Automata Failures : A. Calculus and a Method", IBM Journal of Research and Development, No. 10, Oct. 1966.

[5] Armstrong, D.B., "A Deductive Method for Simulating Faults in Logic Circuits", IEEE TC, Vol. C-22, No. 5, May 1972.

[6] Donath, W.E., "Complexity Theory and Design Automation", 17th DA Conference, June 1980.

[7] Goel, P., "Test generation cost analysis and projections", 17th DA Conference, June 1980.

[8] Williams, T.W., "Survey of Design for Testability", IEEE Transactions on CAD. To appear Jan. 1982.

[9] Williams, T.W. and Parker, K.P., "Testing Logic Networks and Design for Testability". Computer, Oct. 1979.

[10] Grason, J. and Nogle, A., "Digital Test Generation and Design for Testability", 17th DA Conference, June 1980.

[11] Ando, H., "Testing VLSI with Random Access Scan", Digest of Papers Compcon 1980, Feb. 1980.

[12] Kohavi, Z., "Switching and Finite Automata Theory", McGraw-Hill, 1970, pp. 489-540.

[13] Konemann, B., Mucha, J. and Zwiehoff, G., "Built in Logic Block Observation Techniques", IEEE Test Conference, Oct. 1979.