Design for Testability—A Survey

THOMAS W. WILLIAMS, MEMBER, IEEE, AND KENNETH P. PARKER, MEMBER, IEEE

Invited Paper

Abstract—This paper discusses the basics of design for testability. A short review of testing is given along with some reasons why one should test. The different techniques of design for testability are discussed in detail. These include techniques which can be applied to today's technologies and techniques which have been recently introduced and will soon appear in new designs.

I. Introduction

INTEGRATED Circuit Technology is now moving from Large-Scale Integration (LSI) to Very-Large-Scale Integration (VLSI). This increase in gate count, which now can be as much as factors of three to five times, has also brought a decrease in gate costs, along with improvements in performance. All these attributes of VLSI are welcomed by the industry. However, a problem never adequately solved by LSI is still with us and is getting much worse: the problem of determining, in a cost-effective way, whether a component, module, or board has been manufactured correctly [1]-[3], [52]-[68].

The testing problem has two major facets:

1) test generation [74]-[99]
2) test verification [100]-[114].

Test generation is the process of enumerating stimuli for a circuit which will demonstrate its correct operation. Test verification is the process of proving that a set of tests are effective towards this end. To date, formal proof has been impossible in practice. Fault simulation has been our best alternative, yielding a quantitative measure of test effectiveness. With the vast increase in circuit density, the ability to generate test patterns automatically and conduct fault simulation with these patterns has drastically waned. As a result, some manufacturers are foregoing these more rigorous approaches and are accepting the risks of shipping a defective product. One general approach to addressing this problem is embodied in a collection of techniques known as "Design for Testability" [12]-[35].

Design for Testability initially attracted interest in connection with LSI designs. Today, in the context of VLSI, the phrase is gaining even more currency. The collection of techniques that comprise Design for Testability are, in some cases, general guidelines; in other cases, they are hard and fast design rules. Together, they can be regarded essentially as a menu of techniques, each with its associated cost of implementation and return on investment. The purpose of this paper is to present the basic concepts in testing, beginning with the fault models and carrying through to the different techniques associated with Design for Testability which are known today in the public sector. The design for testability techniques are divided into two categories [10]. The first category is that of the ad hoc technique for solving the testing problem. These techniques solve a problem for a given design and are not generally applicable to all designs. This is contrasted with the second category of structured approaches. These techniques are generally applicable and usually involve a set of design rules by which designs are implemented. The objective of a structured approach is to reduce the sequential complexity of a network to aid test generation and test verification.

The first ad hoc approach is partitioning [13], [17], [23], [26]. Partitioning is the ability to disconnect one portion of a network from another portion of a network in order to make testing easier. The next approach which is used at the board level is that of adding extra test points [23], [24]. The third ad hoc approach is that of Bus Architecture Systems [12], [27]. This is similar to the partitioning approach and allows one to divide and conquer—that is, to be able to reduce the network to smaller subnetworks which are much more manageable. These subnetworks are not necessarily designed with any design for testability in mind. The forth technique which bridges both the structured approach and the ad hoc approach is that of Signature Analysis [12], [27], [33], [55]. Signature Analysis requires some design rules at the board level, but is not directed at the same objective as the structure approaches are—that is, the ability to observe and control the state variables of a sequential machine.

For structured approaches, there are essentially four categories which will be discussed—the first of which is a multiplexer technique [14], [21], Random Access Scan, that has been recently published and has been used, to some extent, by others before. The next techniques are those of the Level-Sensitive Scan Design (LSSD) [16], [18]-[20], [34], [35] approach and the Scan Path approach which will be discussed in detail. These techniques allow the test generation problem to be completely reduced to one of generating tests for combinational logic. Another approach which will be discussed is that of the Scan/Set Logic [31]. This is similar to the LSSD approach and the Scan Path approach since shift registers are used to load and unload data. However, these shift registers are not part of the system data path and all system latches are not necessarily controllable and observable via the shift register. The fourth approach which will be discussed is that of Built-In Logic Block Observation (BILBO) [25] which has just recently been proposed. This technique has the attributes of both the LSSD network and Scan Path network, the ability to separate the network into combinational and sequential parts, and has the attribute of Signature Analysis—that is, employing linear feedback shift registers.

For each of the techniques described under the structured approach, the constraints, as well as various ways in which

Manuscript received June 14, 1982; revised September 15, 1982.

T. W. Williams is with IBM, General Technology Division, Boulder, CO 80302.
K. P. Parker is with Hewlett-Packard, Loveland Instrument Division, Loveland, CO 80537.

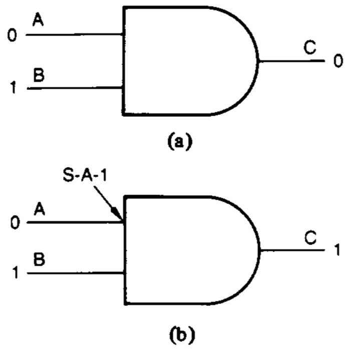


Fig. 1. Test for input stuck at fault. (a) Fault-free AND gate (good machine). (b) Faulty AND gate (faulty machine).

they can be exploited in design, manufacturing, testing, and field servicing will be described. The basic storage devices and the general logic structure resulting from the design constraints will be described in detail. The important question of how much it costs in logic gates and operating speed will be discussed qualitatively. All the structured approaches essentially allow the controllability and observability of the state variables in the sequential machine. In essence, then, test generation and fault simulation can be directed more at a combinational network, rather than at a sequential network.

A. Definitions and Assumptions

A model of faults which is used throughout the industry that does not take into account all possible defects, but is a more global type of model, is the Stuck-At model. The Stuck-At model [1]-[3], [9], [11] assumes that a logic gate input or output is fixed to either a logic 0 or a logic 1. Fig. 1(a) shows an AND gate which is fault-free. Fig. 1(b) shows an AND gate with input "*A*," Stuck-At-1 (S-A-1).

The faulty AND gate perceives the "*A*" input as 1, irrespective of the logic value placed on the input. The pattern applied to the fault-free AND gates in Fig. 1 has an output value of 0 since the input is 0 on the "*A*" input and 1 on the "*B*" input, and the AND'ing of those two leads to a 0 on the output. The pattern in Fig. 1(b) shows an output of 1, since the "*A*" input is perceived as a 1 even though a 0 is applied to that input. The 1 on the "*B*" input is perceived as a 1, and the results are AND'ed together to give a 1 output. Therefore, the pattern shown in Fig. 1(a) and (b) is a test for the "*A*" input, S-A-1, since there is a difference between the faulty gate (faulty machine) and the good gate (good machine). This pattern 01 on the "*A*" and "*B*" inputs, respectively, is considered a test because the good machine responds differently from the faulty machine. If they had the same response then that pattern would not have constituted a test for that fault.

If a network contained N nets, any net may be good, Stuck-At 1 or Stuck-At 0; thus all possible network state combinations would be 3^N. A network with 100 nets, then, would contain 5×10^{47} different combinations of faults. This would be far too many faults to assume. The run time of any program trying to generate tests or fault simulate tests for this kind of design would be impractical.

Therefore, the industry, for many years, has clung to the single Stuck-At fault assumption. That is, a good machine will have no faults. The faulty machines that are assumed will have one, and only one, of the stuck faults. In other words, all faults taken two at a time are not assumed, nor are all faults taken three at a time, etc. History has proven that the single Stuck-At fault assumption, in prior technologies, has been adequate. However, there could be some problems in LSI—particularly with CMOS using the single Stuck-At fault assumption.

The problem with CMOS is that there are a number of faults which could change a combinational network into a sequential network. Therefore, the combinational patterns are no longer effective in testing the network in all cases. It still remains to be seen whether, in fact, the single Stuck-At fault assumption will survive the CMOS problems.

Also, the single Stuck-At fault assumption does not, in general, cover the bridging faults [43] that may occur. Historically again, bridging faults have been detected by having a high level—that is, in the high 90 percent—single Stuck-At fault coverage, where the single Stuck-At fault coverage is defined to be the number of faults that are tested divided by the number of faults that are assumed.

B. The VLSI Testing Problem

The VLSI testing problem is the sum of a number of problems. All the problems, in the final analysis, relate to the cost of doing business (dealt with in the following section). There are two basic problem areas:

1) test generation
2) test verification via fault simulation.

With respect to test generation, the problem is that as logic networks get larger, the ability to generate tests automatically is becoming more and more difficult.

The second facet of the VLSI testing problem is the difficulty in fault simulating the test patterns. Fault simulation is that process by which the fault coverage is determined for a specific set of input test patterns. In particular, at the conclusion of the fault simulation, every fault that is detected by the given pattern set is listed. For a given logic network with 1000 two-input logic gates, the maximum number of single Stuck-At faults which can be assumed is 6000. Some reduction in the number of single Stuck-At faults can be achieved by fault equivalencing [36], [38], [41], [42], [47]. However, the number of single Stuck-At faults needed to be assumed is about 3000. Fault simulation, then, is the process of applying every given test pattern to a fault-free machine and to each of the 3000 copies of the good machine containing one, and only one, of the single Stuck-At faults. Thus fault simulation, with respect to run time, is similar to doing 3001 good machine simulations.

Techniques are available to reduce the complexity of fault simulation, however, it still is a very time-consuming, and hence, expensive task [96], [104], [105], [107], [110], [112]-[114].

It has been observed that the computer run time to do test [80] generation and fault simulation is approximately proportional to the number of logic gates to the power of 3;[1] hence, small increases in gate count will yield quickly increasing run times. Equation (1)

[1]The value of the exponent given here (3) is perhaps pessimistic in some cases. Other analyses have used the value 2 instead. A quick rationale goes as follows: with a linear increase k in circuit size comes an attendant linear increase in the number of failure mechanisms (now yielding k squared increase in work). Also, as circuits become larger, they tend to become more strongly connected such that a given block is effected by more blocks and even itself. This causes more work to be done in a range we feel to be k cubed. This fairly nebulous concept of connectivity seems to be the cause for debate on whether the exponent should be 3 or some other value.

$$T = KN^3 \tag{1}$$

shows this relationship, where T is computer run time, N is the number of gates, and K is the proportionality constant. The relationship does not take into account the falloff in automatic test generation capability due to sequential complexity of the network. It has been observed that computer run time just for fault simulation is proportional to N^2 without even considering the test generation phase.

When one talks about testing, the topic of functional testing always comes up as a feasible way to test a network. Theoretically, to do a complete functional test ("exhaustive" testing) seems to imply that all entries in a Karnaugh map (or excitation table) must be tested for a 1 or a 0. This means that if a network has N inputs and is purely combinational, then 2^N patterns are required to do a complete functional test. Furthermore, if a network has N inputs with M latches, at a minimum it takes 2^{N+M} patterns to do a complete functional test. Rarely is that minimum ever obtainable; and in fact, the number of tests required to do a complete functional test is very much higher than that. With LSI, this may be a network with $N = 25$ and $M = 50$, or 2^{75} patterns, which is approximately 3.8×10^{22} Assuming one had the patterns and applied them at an application rate of 1 μs per pattern, the test time would be over a billion years (10^9).

C. *Cost of Testing*

One might ask why so much attention is now being given to the level of testability at chip and board levels. The bottom line is the cost of doing business. A standard among people familiar with the testing process is: If it costs $0.30 to detect a fault at the chip level, then it would cost $3 to detect that same fault when it was embedded at the board level; $30 when it is embedded at the system level; and $300 when it is embedded at the system level but has to be found in the field. Thus if a fault can be detected at a chip or board level, then significantly larger costs per fault can be avoided at subsequent levels of packaging.

With VLSI and the inadequacy of automatic test generation and fault simulation, there is considerable difficulty in obtaining a level of testability required to achieve acceptable defect levels. If the defect level of boards is too high, the cost of field repairs is also too high. These costs, and in some cases, the inability to obtain a sufficient test, have led to the need to have "Design for Testability."

II. Design for Testability

There are two key concepts in Design for Testability: controllability and observability. Control and observation of a network are central to implementing its test procedure. For example, consider the case of the simple AND block in Fig. 1. In order to be able to test the "A" input Stuck-At 1, it was necessary to control the "A" input to 0 and the "B" input to 1 and be able to observe the "C" output to determine whether a 0 was observed or a 1 was observed. The 0 is the result of the good machine, and the 1 would be the result, if you had a faulty machine. If this AND block is embedded into a much larger sequential network, the requirement of being able to control the "A" and "B" inputs to 0 and 1, respectively, and being able to observe the output "C," be it through some other logic blocks, still remains. Therein lies part of the problem of being able to generate tests for a network.

Because of the need to determine if a network has the attributes of controllability and observability that are desired, a number of programs have been written which essentially give analytic measures of controllability and observability for different nets in a given sequential network [69]-[73].

After observing the results of one of these programs in a given network, the logic designer can then determine whether some of the techniques, which will be described later, can be applied to this network to ease the testing problem. For example, test points may be added at critical points which are not observable or which are not controllable, or some of the techniques of Scan Path or LSSD can be used to initialize certain latches in the machine to avoid the difficulties of controllability associated with sequential machines. The popularity of such tools is continuing to grow, and a number of companies are now embarking upon their own controllability/observability measures.

III. Ad Hoc Design for Testability [10]

Testing has moved from the afterthought position that it used to occupy to part of the design environment in LSI and VLSI. When testing was part of the afterthought, it was a very expensive process. Products were discarded because there was no adequate way to test them in production quantities.

There are two basic approaches which are prevalent today in the industry to help solve the testing problem. The first approach categorized here is Ad Hoc, and the second approach is categorized as a Structured Approach. The Ad Hoc techniques are those techniques which can be applied to a given product, but are not directed at solving the general sequential problem. They usually do offer relief, and their cost is probably lower than the cost of the Structured Approaches. The Structured Approaches, on the other hand, are trying to solve the general problem with a design methodology, such that when the designer has completed his design from one of these particular approaches, the results will be test generation and fault simulation at acceptable costs. Structured Approaches lend themselves more easily to design automation. Again, the main difference between the two approaches is probably the cost of implementation and hence, the return on investment for this extra cost. In the Ad Hoc approaches, the job of doing test generation and fault simulation are usually not as simple or as straightforward as they would be with the Structured Approaches, as we shall see shortly.

A number of techniques have evolved from MSI to LSI and now into VLSI that fall under the category of the ad hoc approaches of "Design for Testability." These techniques are usually solved at the board level and do not necessarily require changes in the logic design in order to accomplish them.

A. *Partitioning*

Because the task of test pattern generation and fault simulation is proportional to the number of logic gates to the third power, a significant amount of effort has been directed at approaches called "Divide and Conquer."

There are a number of ways in which the partitioning approach to Design for Testability can be implemented. The first is to mechanical partition by dividing a network in half. In essence, this would reduce the test generation and fault simulation tasks by 8 for two boards. Unfortunately, having two boards rather than one board can be a significant cost disadvantage and defeats the purpose of integration.

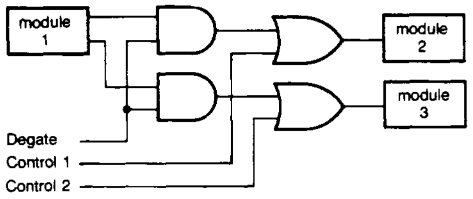


Fig. 2. Use of degating logic for logical partioning.

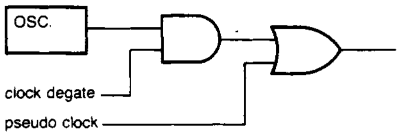


Fig. 3. Degating lines for oscillator.

Another approach that helps the partitioning problem, as well as helping one to "Divide and Conquer" is to use jumper wires. These wires would go off the board and then back on the board, so that the tester and the test generator can control and observe these nets directly. However, this could mean a significant number of I/O contacts at the board level which could also get very costly.

Degating is another technique for separating modules on a board. For example, in Fig. 2, a degating line goes to two AND blocks that are driven from Module 1. The results of those two AND blocks go to two independent OR blocks—one controlled by Control Line 1, the other with Control Line 2. The output of the OR block from Control Line 1 goes into Module 2, and the output of Control Line 2 goes into Module 3. When the degate line is at the 0 value, the two Control Lines, 1 and 2, can be used to drive directly into Modules 2 and 3. Therefore, complete controllability of the inputs to Modules 2 and 3 can be obtained by using these control lines. If those two nets happen to be very difficult nets to control, as pointed out, say, by a testability measure program, then this would be a very cost-effective way of controlling those two nets and hence, being able to derive the tests at a very reasonable cost.

A classical example of degating logic is that associated with an oscillator, as shown in Fig. 3. In general, if an oscillator is free-running on a board, driving logic, it is very difficult, and sometimes impossible, to synchronize the tester with the activity of the logic board. As a result, degating logic can be used here to block the oscillator and have a pseudo-clock line which can be controlled by the tester, so that the dc testing of all the logic on that board can be synchronized. All of these techniques require a number of extra primary inputs and primary outputs and possibly extra modules to perform the degating.

B. Test Points

Another approach to help the controllability and observability of a sequential network is to use test points [23], [24]. If a test point is used as a primary input to the network, then that can function to enhance controllability. If a test point is used as a primary output, then that is used to enhance the observability of a network. In some cases, a single pin can be used as both an input and an output.

For example, in Fig. 4, Module 1 has a degate function, so that the output of those two pins on the module could go to noncontrolling values. Thus the external pins which are dotted into those nets could control those nets and drive Module 2.

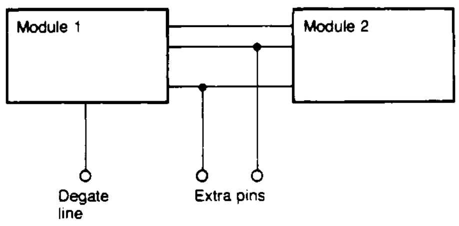


Fig. 4. Test points used as both inputs and outputs.

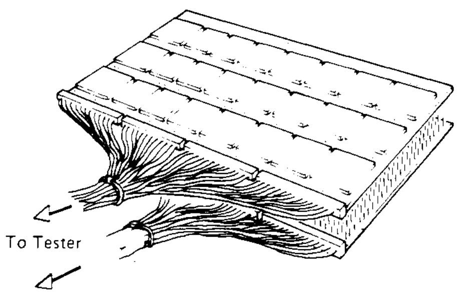


Fig. 5. "Bed of Nails" test.

On the other hand, if the degate function is at the opposite value, then the output of Module 1 can be observed on these external pins. Thus the enhancement of controllability and observability can be accommodated by adding pins which can act as both inputs and outputs under certain degating conditions.

Another technique which can be used for controllability is to have a pin which, in one mode, implies system operation, and in another mode takes N inputs and gates them to a decoder. The 2^N outputs of the decoder are used to control certain nets to values which otherwise would be difficult to obtain. By so doing, the controllability of the network is enhanced.

As mentioned before, predictability is an issue which is as important as controllability and observability. Again, test points can be used here. For example, a CLEAR or PRESET function for all memory elements can be used. Thus the sequential machine can be put into a known state with very few patterns.

Another technique which falls into the category of test points and is very widely used is that of the "Bed of Nails" [31] tester, Fig. 5. The Bed of Nails tester probes the underside of a board to give a larger number of points for observability and controllability. This is in addition to the normal tester contact to the board under test. The drawback of this technique is that the tester must have enough test points to be able to control and observe each one of these nails on the Bed of Nails tester. Also, there are extra loads which are placed on the nets and this can cause some drive and receive problems. Furthermore, the mechanical fixture which will hold the Bed of Nails has to be constructed, so that the normal forces on the probes are sufficient to guarantee reliable contacts. Another application for the Bed of Nails testing is to do "drive/sense nails" [31] or "*in situ*" or "in-circuit" testing, which, effectively, is the technique of testing each chip on the board independently of the other chips on the board. For each chip, the appropriate nails and/or primary inputs are driven so as to prevent one chip from being driven by the other chips on the board. Once this state has been established, the isolated chip on the board can now be tested. In this case, the resolution to the failing

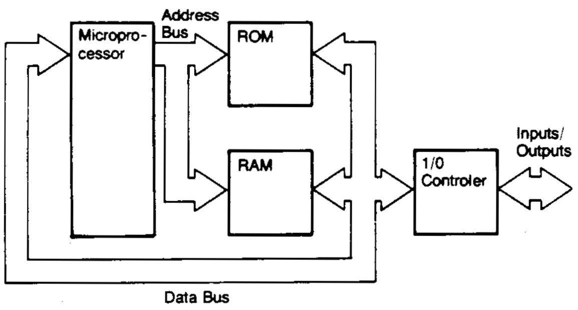


Fig. 6. Bus structured microcomputer.

chip is much better than edge connector tests, however, there is some exposure to incomplete testing of interconnections and care must be taken not to damage the circuit when overdriving it. Design for testability in a Bed of Nails environment must take the issues of contact reliability, multiplicity, and electrical loading into account.

C. Bus Architecture

An approach that has been used very successfully to attack the partitioning problem by the microcomputer designers is to use a bus structured architecture. This architecture allows access to critical buses which go to many different modules on the computer board. For example, in Fig. 6, you can see that the data bus is involved with both the microprocessor module, the ROM module, the RAM module, and the I/O Controller module. If there is external access to the data bus and three of the four modules can be turned off the data bus—that is, their outputs can be put into a high-impedence state (three-state driver)—then the data bus could be used to drive the fourth module, as if it were a primary input (or primary output) to that particular module. Similarly, with the address bus, access again must be controlled externally to the board, and thus the address bus can be very useful to controlling test patterns to the microcomputer board. These buses, in essence, partition the board in a unique way, so that testing of subunits can be accomplished. A drawback of bus-structured designs comes with faults on the bus itself. If a bus wire is stuck, any module or the bus trace itself may be the culprit. Normal testing is done by deducing the location of a fault from voltage information. Isolating a bus failure may require current measurements, which are much more difficult to do.

D. Signature Analysis

This technique for testing, introduced in 1977 [27], [33], [55] is heavily reliant on planning done in the design stage. That is why this technique falls between the Ad Hoc and the Structured Approaches for Design for Testability, since some care must be taken at the board level in order to ensure proper operation of this Signature Analysis of the board [12]. Signature Analysis is well-suited to bus structure architectures, as previously mentioned and in particular, those associated with microcomputers. This will become more apparent shortly.

The integral part of the Signature Analysis approach is that of a linear feedback shift register [8]. Fig. 7 shows an example of a 3-bit linear feedback shift register. This linear feedback shift register is made up of three shift register latches. Each one is represented by a combination of an $L1$ latch and an $L2$ latch. These can be thought of as the master latch being the $L1$ latch and the slave latch being the $L2$ latch. An "A" clock clocks all the $L1$ latches, and a "B" clock clocks all the $L2$ latches, so that turning the "A" and "B" clocks on and off independently will shift the shift register 1-bit position to the right. Furthermore, this linear shift register has an EXCLUSIVE-OR gate which takes the output, $Q2$, the second bit in the shift register, and EXCLUSIVE-OR's it with the third bit in the shift register, $Q3$. The result of that EXCLUSIVE-OR is the input to the first shift register. A single clock could be used for this shift register, which is generally the case, however, this concept will be used shortly when some of the structured design approaches are discussed which use two nonoverlapping clocks. Fig. 7 shows how this linear feedback shift register will count for different initial values.

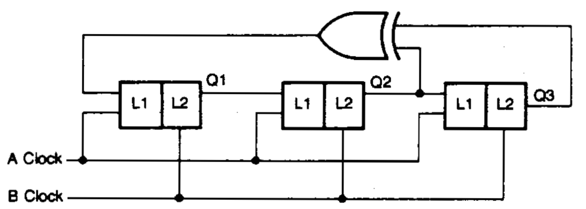


Fig. 7. Counting capabilities of a linear feedback shift register.

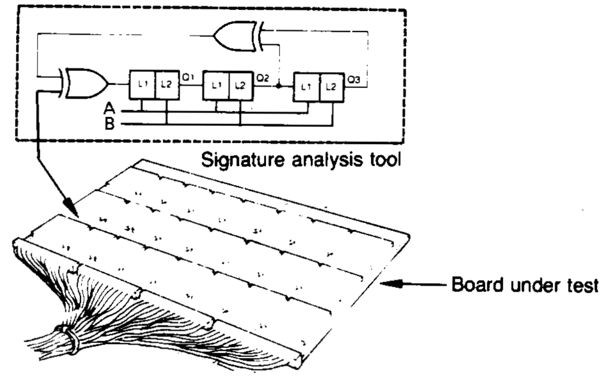


Fig. 8. Use of signature analysis tool.

For longer shift registers, the maximal length linear feedback configurations can be obtained by consulting tables [8] to determine where to tap off the linear feedback shift register to perform the EXCLUSIVE-OR function. Of course, only EXCLUSIVE-OR blocks can be used, otherwise, the linearity would not be preserved.

The key to Signature Analysis is to design a network which can stimulate itself. A good example of such a network would be microprocessor-based boards, since they can stimulate themselves using the intelligence of the processor driven by the memory on the board.

The Signature Analysis procedure is one which has the shift register in the Signature Analysis tool, which is external to the board and not part of the board in any way, synchronized with the clocking that occurs on the board, see Fig. 8. A probe is used to probe a particular net on the board. The result of that probe is EXCLUSIVE-OR'ed into the linear feedback shift register. Of course, it is important that the linear feedback shift register be initialized to the same starting place every time, and that the clocking sequence be a fixed number, so that the tests can be repeated. The board must also have some initialization, so that its response will be repeated as well.

After a fixed number of clock periods—let's assume 50—a particular value will be stored in $Q1$, $Q2$, and $Q3$. It is not necessarily the value that would have occurred if the linear feedback shift register was just counted 50 times—Modulo 7.

The value will be changed, because the values coming from the board via the probe will not necessarily be a continuous string of 1's; there will be 1's intermixed with 0's.

The place where the shift register stops on the Signature Analysis Tool–that is, the values for $Q1$, $Q2$, and $Q3$ is the Signature for that particular node for the good machine. The question is: If there were errors present at one or more points in the string of 50 observations of that particular net of the board, would the value stored in the shift register for $Q1$, $Q2$, and $Q3$ be different than the one for the good machine? It has been shown that with a 16-bit linear feedback shift register, the probability of detecting one or more errors is extremely high [55]. In essence, the signature, or "residue," is the remainder of the data stream after division by an irreduceable polynomial. There is considerable data compression–that is, after the results of a number of shifting operations, the test data are reduced to 16 bits, or, in the case of Fig. 8, 3 bits. Thus the result of the Signature Analysis tool is basically a Go/No-Go for the output for that particular module.

If the bad output for that module were allowed to cycle around through a number of other modules on the board and then feed back into this particular module, it would not be clear after examining all the nodes in the loop which module was defective–whether it was the module whose output was being observed, or whether it was another module upstream in the path. This gives rise to two requirements for Signature Analysis. First of all, closed-loop paths must be broken at the board level. Second, the best place to start probing with Signature Analysis is with a "kernel" of logic. In other words, on a microprocessor-based board, one would start with the outputs of the microprocessor itself and then build up from that particular point, once it has been determined that the microprocessor is good.

This breaking of closed loops is a tenant of Design for Testability and for Signature Analysis. There is a little overhead for implementing Signature Analysis. Some ROM space would be required (to stimulate the self-test), as well as extra jumpers, in order to break closed loops on the board. Once this is done, however, the test can be obtained for very little cost. The only question that remains is about the quality of the tests–that is, how good are the tests that are being generated, do they cover all the faults, etc.

Unfortunately, the logic models–for example, microprocessors–are not readily available to the board user. Even if a microprocessor logic model were available, they would not be able to do a complete fault simulation of the patterns because it would be too large. Hence, Signature Analysis may be the best that could be done for this particular board with the given inputs which the designer has. Presently, large numbers of users are currently using the Signature Analysis technique to test boards containing LSI and VLSI components.

IV. Structured Design for Testability

Today, with the utilization of LSI and VLSI technology, it has become apparent that even more care will have to be taken in the design stage in order to ensure testability and produceability of digital networks. This has led to rigorous and highly structured design practices. These efforts are being spearheaded not by the makers of LSI/VLSI devices but by electronics firms which possess captive IC facilities and the manufacturers of large main-frame computers.

Most structured design practices [14]-[16], [18]-[21], [25], [31], [32], [34], [35] are built upon the concept that if the values in all the latches can be controlled to any specific value, and if they can be observed with a very straightforward operation then the test generation, and possibly, the fault task, can be reduced to that of doing test generation and fault simulation for a combinational logic network. A control signal can switch the memory elements from their normal mode of operation to a mode that makes them controllable and observable.

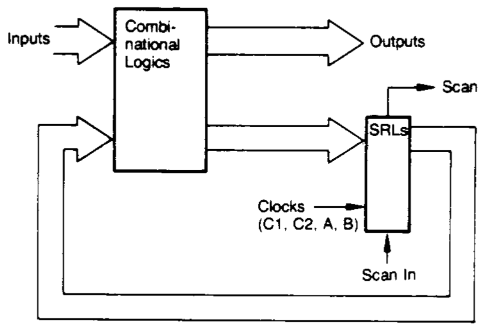


Fig. 9. Classical model of a sequential network utilizing a shift register for storage.

It appears from the literature that several companies, such as IBM, Fujitsu Ltd., Sperry-Univac, and Nippon Electric Co., Ltd. [14]-[16], [18]-[21], [31], [32], [35] have been dedicating formidable amounts of resources toward Structured Design for Testability. One notes simply by scanning the literature on testing, that many of the practical concepts and tools for testing were developed by main-frame manufacturers who do not lack for processor power. It is significant, then, that these companies, with their resources, have recognized that unstructured designs lead to unacceptable testing problems. Presently, IBM has extensively documented its efforts in Structured Design for Testability, and these are reviewed first.

A. Level-Sensitive Scan Design (LSSD)

With the concept that the memory elements in an IC can be threaded together into a shift register, the memory elements values can be both controlled and observed. Fig. 9 shows the familiar generalized sequential circuit model modified to use a shift register. This technique enhances both controllability and observability, allowing us to augment testing by controlling inputs and internal states, and easily examining internal state behavior. An apparent disadvantage is the serialization of the test, potentially costing more time for actually running a test.

LSSD is IBM's discipline for structural design for testability. "Scan" refers to the ability to shift into or out of any state of the network. "Level-sensitive" refers to constraints on circuit excitation, logic depth, and the handling of clocked circuitry. A key element in the design is the "shift register latch" (SRL) such as can be implemented in Fig. 10. Such a circuit is immune to most anomalies in the ac characteristics of the clock, requiring only that it remain high (sample) at least long enough to stabilize the feedback loop, before being returned to the low (hold) state [18], [19]. The lines D and C form the normal mode memory function while lines I, A, B, and $L2$ comprise additional circuitry for the shift register function.

The shift registers are threaded by connecting I to $L2$ and operated by clocking lines A and B in two-phase fashion. Fig. 11 shows four modules threaded for shift register action. Now note in Fig. 11 that each module could be an SRL or, one level up, a board containing threaded IC's, etc. Each level of pack-

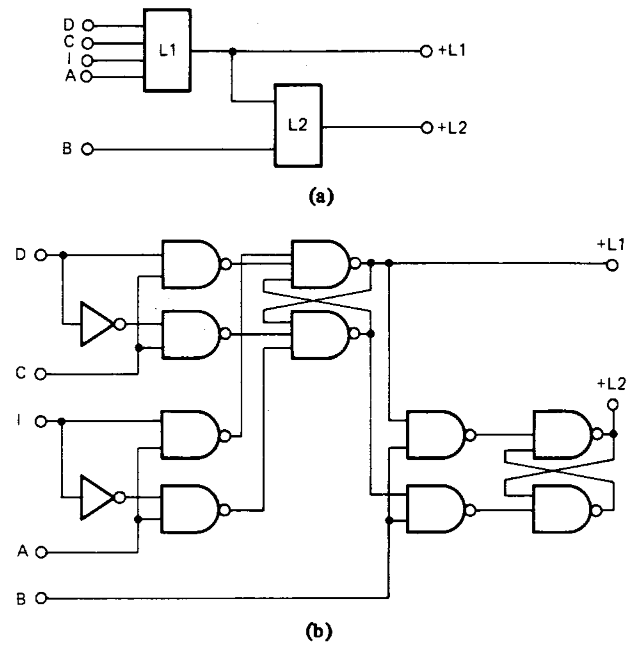


Fig. 10. Shift register latch (SRL). (a) Symbolic representation. (b) Implementation in AND-INVERT gates.

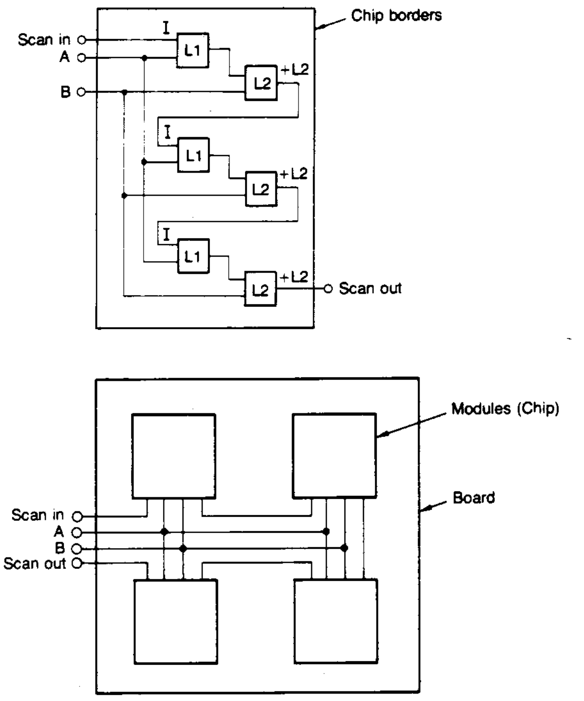


Fig. 11. Interconnection of SRL's on an integrated circuit and board.

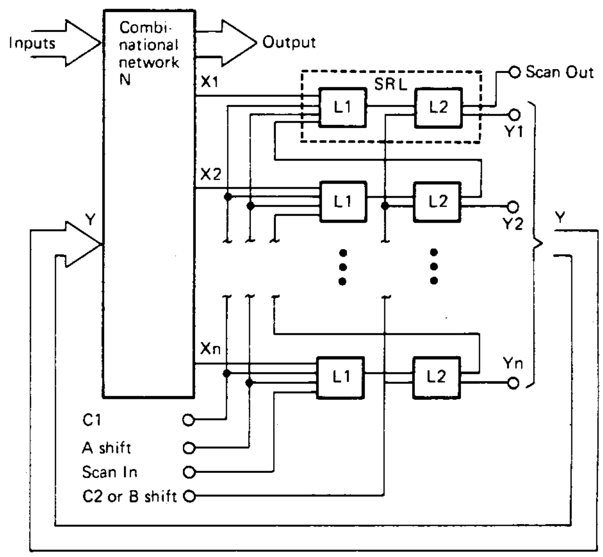


Fig. 12. General structure of an LSSD subsystem with two system clocks.

aging requires the same four additional lines to implement the shift register scan feature. Fig. 12 depicts a general structure for an LSSD subsystem with a two-phase system clock. Additional rules concerning the gating of clocks, etc., are given by Williams and Eichelberger [18], [19]. Also, it is not practical to implement RAM with SRL memory, so additional procedures are required to handle embedded RAM circuitry [20].

Given that an LSSD structure is achieved, what are the rewards? It turns out that the network can now be thought of as purely combinational, where tests are applied via primary inputs and shift-register outputs. The testing of combinational circuits is a well understood and (barely) tractable problem. Now techniques such as the D-Algorithm [93] compiled code Boolean simulation [2], [74], [106], [107], and adaptive random test generation [87], [95], [98] are again viable approaches to the testing problem. Further, as small subsystems are tested, their aggregates into larger systems are also testable by cataloging the position of each testable subsystem in the shift register chain. System tests become (ideally) simple concatenations of subsystem tests. Though ideals are rarely achieved, the potential for solving otherwise hopeless testing problems is very encouraging.

In considering the cost performance impacts, there are a number of negative impacts associated with the LSSD design philosophy. First of all, the shift register latches in the shift register are, logically, two or three times as complex as simple latches. Up to four additional primary inputs/outputs are required at each package level for control of the shift registers. External asynchronous input signals must not change more than once every clock cycle. Finally, all timing within the subsystem is controlled by externally generated clock signals.

In terms of additional complexity of the shift register hold latches, the overhead from experience has been in the range of 4 to 20 percent. The difference is due to the extent to which the system designer made use of the $L2$ latches for system function. It has been reported in the IBM System 38 literature that 85 percent of the $L2$ latches were used for system function. This drastically reduces the overhead associated with this design technique.

With respect to the primary inputs/outputs that are required to operate the shift register, this can be reduced significantly by making functional use of some of the pins. For example, the scan-out pin could be a functional output of an SRL for that particular chip. Also, overall performance of the subsystem may be degraded by the clocking requirement, but the effect should be small.

The LSSD structured design approach for Design for Testability eliminates or alleviates some of the problems in designing, manufacturing and maintaining LSI systems at a reasonable cost.

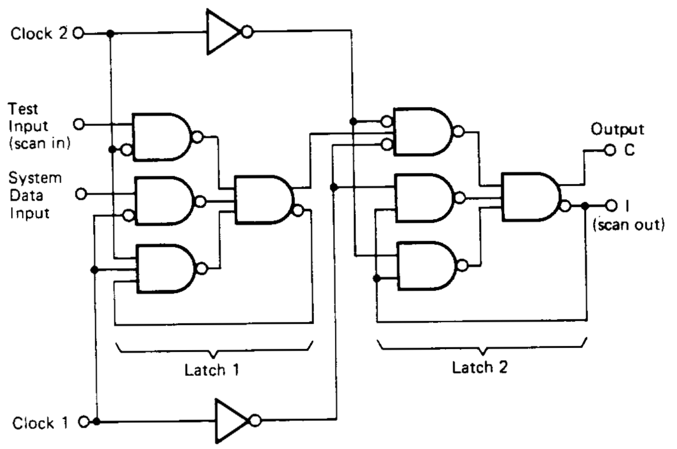


Fig. 13. Raceless D-type flip-flop with Scan Path.

B. Scan Path

In 1975, a survey paper of test generation systems in Japan was presented by members of Nippon Electric Co., Ltd. [21]. In that survey paper, a technique they described as Scan Path was presented. The Scan Path technique has the same objectives as the LSSD approach which has just been described. The Scan Path technique similarities and differences to the LSSD approach will be presented.

The memory elements that are used in the Scan Path approach are shown in Fig. 13. This memory element is called a raceless D-type flip-flop with Scan Path.

In system operation, Clock 2 is at a logic value of 1 for the entire period. This, in essence, blocks the test or scan input from affecting the values in the first latch. This D-type flip-flop really contains two latches. Also, by having Clock 2 at a logic value of 1, the values in Latch 2 are not disturbed.

Clock 1 is the sole clock in system operation for this D-type flip-flop. When Clock 1 is at a value of 0, the System Data Input can be loaded into Latch 1. As long as Clock 1 is 0 for sufficient time to latch up the data, it can then turn off. As it turns off, it then will make Latch 2 sensitive to the data output of Latch 1. As long as Clock 1 is equal to a 1 so that data can be latched up into Latch 2, reliable operation will occur. This assumes that as long as the output of Latch 2 does not come around and feed the system data input to Latch 1 and change it during the time that the inputs to both Latch 1 and Latch 2 are active. The period of time that this can occur is related to the delay of the inverter block for Clock 1. A similar phenomenon will occur with Clock 2 and its associated inverter block. This race condition is the exposure to the use of only one system clock.

This points out a significant difference between the Scan Path approach and the LSSD approach. One of the basic principles of the LSSD approach is level-sensitive operation—the ability to operate the clocks in such a fashion that no races will exist. In the LSSD approach, a separate clock is required for Latch 1 from the clock that operates Latch 2.

In terms of the scanning function, the D-type flip-flop with Scan Path has its own scan input called test input. This is clocked into the $L1$ latch by Clock 2 when Clock 2 is a 0, and the results of the $L1$ latch are clocked into Latch 2 when Clock 2 is a 1. Again, this applies to master/slave operation of Latch 1 and Latch 2 with its associated race with proper attention to delays this race will not be a problem.

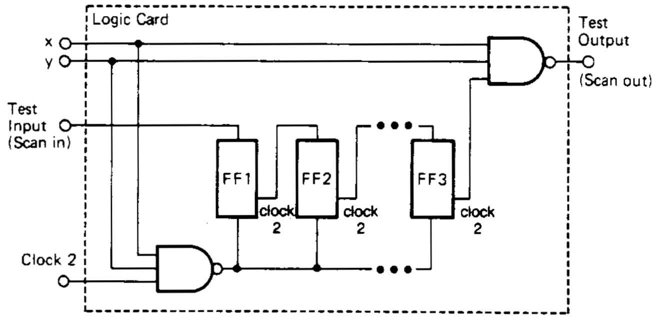


Fig. 14. Configuration of Scan Path on Card.

Another feature of the Scan Path approach is the configuration used at the logic card level. Modules on the logic card are all connected up into a serial scan path, such that for each card, there is one scan path. In addition, there are gates for selecting a particular card in a subsystem. In Fig. 14, when X and Y are both equal to 1—that is the selection mechanism—Clock 2 will then be allowed to shift data through the scan path. Any other time, Clock 2 will be blocked, and its output will be blocked. The reason for blocking the output is that a number of card outputs can then be put together; thus the blocking function will put their output to noncontrolling values, so that a particular card can have unique control of the unique test output for that system.

It has been reported by the Nippon Electric Company that they have used the Scan Path approach, plus partitioning which will be described next, for systems with 100 000 blocks or more. This was for the FLT-700 System, which is a large processor system.

The partitioning technique is one which automatically separates the combinational network into smaller subnetworks, so that the test generator can do test generation for the small subnetworks, rather than the larger networks. A partition is automatically generated by backtracing from the D-type flip-flops, through the combinational logic, until it encounters a D-type flip-flop in the backtrace (or primary input). Some care must be taken so that the partitions do not get too large.

To that end, the Nippon Electric Company approach has used a controlled D-type flip-flop to block the backtracing of certain partitions when they become too high. This is another facet of Design for Testability—that is, the introduction of extra flip-flops totally independent of function, in order to control the partitioning algorithm.

Other than the lack of the level sensitive attribute to the Scan Path approach, the technique is very similar to the LSSD approach. The introduction of the Scan Path approach was the first practical implementation of shift registers for testing which was incorporated in a total system.

C. Scan/Set Logic

A technique similar to Scan Path and LSSD, but not exactly the same, is the Scan/Set technique put forth by Sperry-Univac [31]. The basic concept of this technique is to have shift registers, as in Scan Path or in LSSD, but these shift registers are not in the data path. That is, they are not in the system data path; they are independent of all the system latches. Fig. 15 shows an example of the Scan/Set Logic, referred to as bit serial logic.

The basic concept is that the sequential network can be

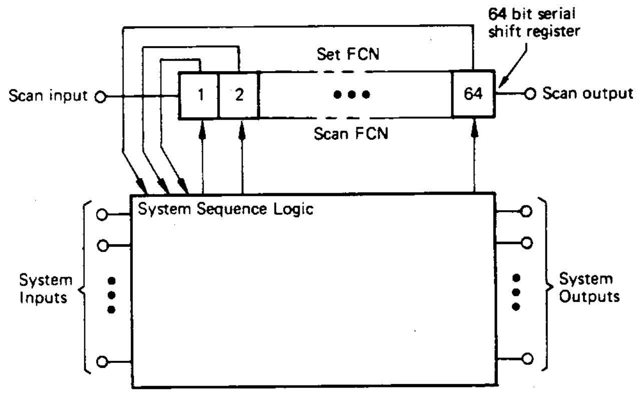


Fig. 15. Scan/Set Logic (bit-serial).

sampled at up to 64 points. These points can be loaded into the 64-bit shift register with a single clock. Once the 64 bits are loaded, a shifting process will occur, and the data will be scanned out through the scan-out pin. In the case of the set function, the 64 bits can be funneled into the system logic, and then the appropriate clocking structure required to load data into the system latches is required in this system logic. Furthermore, the set function could also be used to control different paths to ease the testing function.

In general, this serial Scan/Set Logic would be integrated onto the same chip that contrains sequential system logic. However, some applications have been put forth where the bit serial Scan/Set Logic was off-chip, and the bit-serial Scan/Set Logic only sampled outputs or drove inputs to facilitate in-circuit testing.

Recently, Motorola has come forth with a chip which is T^2L and which has I^2L logic integrated on that same chip. This has the Scan/Set Logic bit serial shift registers built in I^2L. The T^2L portion of the chip is a gate array, and the I^2L is on the chip, whether the customer wants it or not. It is up to the customer to use the bit-serial logic if he chooses.

At this point, it should be explained that if all the latches within the system sequential network are not both scanned and set, then the test generation function is not necessarily reduced to a total combinational test generation function and fault simulation function. However, this technique will greatly reduce the task of test generation and fault simulation.

Again, the Scan/Set technique has the same objectives as Scan Path and LSSD—that is, controllability and observability. However, in terms of its implementation, it is not required that the set function set all system latches, or that the scan function scan all system latches. This design flexibility would have a reflection in the software support required to implement such a technique.

Another advantage of this technique is that the scan function can occur during system operation—that is, the sampling pulse to the 64-bit serial shift register can occur while system clocks are being applied to the system sequential logic, so that a snapshot of the sequential machine can be obtained and off-loaded without any degradation in system performance.

D. Random-Access Scan

Another technique similar to the Scan Path technique and LSSD is the Random-Access Scan technique put forth by Fujitsu [14]. This technique has the same objective as Scan Path and LSSD—that is, to have complete controllability and observability of all internal latches. Thus the test generation function can be reduced to that of combinational test generation and combinational fault simulation as well.

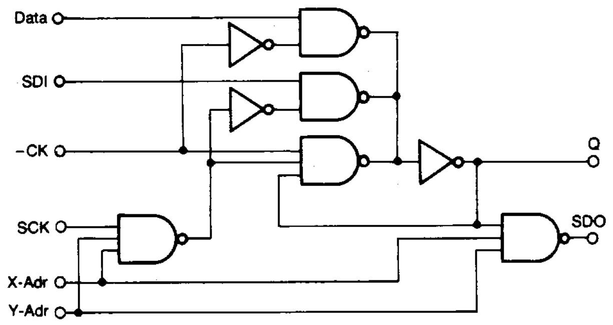


Fig. 16. Polarity-hold-type addressable latch.

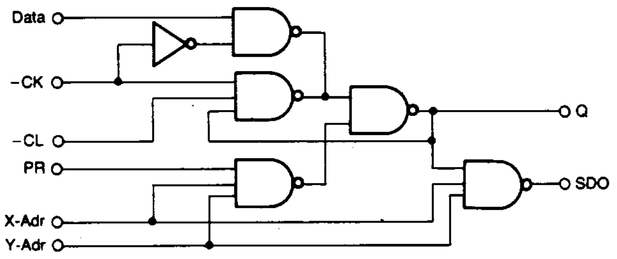


Fig. 17. Set/Reset type addressable latch.

Random-Access Scan differs from the other two techniques in that shift registers are not employed. What is employed is an addressing scheme which allows each latch to be uniquely selected, so that it can be either controlled or observed. The mechanism for addressing is very similar to that of a Random-Access Memory, and hence, its name.

Figs. 16 and 17 show the two basic latch configurations that are required for the Random-Access Scan approach. Fig. 16 is a single latch which has added to it an extra data port which is a Scan Data In port (SDI). These data are clocked into the latch by the SCK clock. The SCK clock can only affect this latch, if both the X and Y addresses are one. Furthermore, when the X address and Y address are one, then the Scan Data Out (SDO) point can be observed. System data labeled Data in Figs. 16 and 17 are loaded into this latch by the system clock labeled CK.

The set/reset-type addressable latch in Fig. 17 does not have a scan clock to load data into the system latch. This latch is first cleared by the CL line, and the CL line is connected to other latches that are also set/reset-type addressable latches. This, then, places the output value Q to a 0 value. A preset is directed at those latches that are required to be set to a 1 for that particular test. This preset is directed by addressing each one of those latches and applying the preset pulse labeled PR. The output of the latch Q will then go to a 1. The observability mechanism for Scan Data Out is exactly the same as for the latch shown in Fig. 16.

Fig. 18 gives an overall view of the system configuration of the Random-Access Scan approach. Notice that, basically, there is a Y address, an X address, a decoder, the addressable storage elements, which are the memory elements or latches, and the sequential machine, system clocks, and CLEAR function. There is also an SDI which is the input for a given latch, an SDO which is the output data for that given latch, and a scan clock. There is also one logic gate necessary to create the preset function.

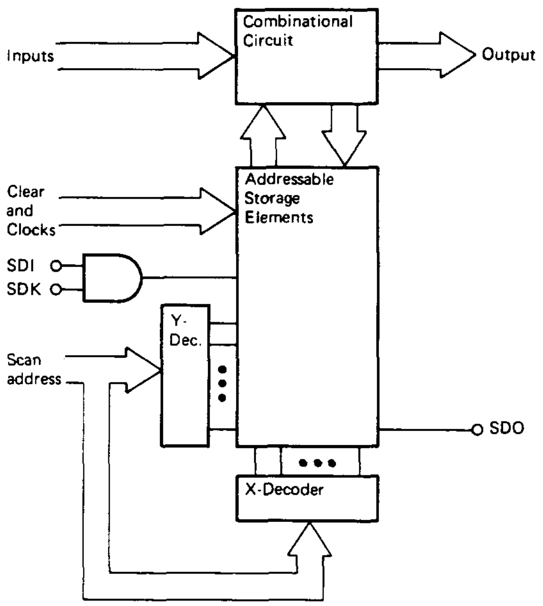


Fig. 18. Random-Access Scan network.

The Random-Access Scan technique allows the observability and controllability of all system latches. In addition, any point in the combinational network can be observed with the addition of one gate per observation point, as well as one address in the address gate, per observation point.

While the Scan Path approach and the LSSD approach require two latches for every point which needs to be observed, the overhead for Random-Access Scan is about three to four gates per storage element. In terms of primary inputs/outputs, the overhead is between 10 and 20. This pin overhead can be diminished by using the serial scan approach for the X and Y address counter, which would lead to 6 primary inputs/outputs.

V. Self-Testing and Built-In Tests

As a natural outgrowth of the Structured Design approach for "Design for Testability," Self-Tests and Built-In Tests have been getting considerably more attention. Four techniques will be discussed, which fall into this category, BILBO, Syndrome Testing, Testing by Verifying Walsh Testing Coefficients, and Autonomous Testing. Each of these techniques will be described.

A. Built-In Logic Block Observation, BILBO

A technique recently presented takes the Scan Path and LSSD concept and integrates it with the Signature Analysis concept. The end result is a technique for Built-In Logic Block Observation, BILBO [25].

Fig. 19 gives the form of an 8-bit BILBO register. The block labeled L_i ($i = 1, 2, \cdots, 8$) are the system latches. B_1 and B_2 are control values for controlling the different functions that the BILBO register can perform. S_{IN} is the scan-in input to the 8-bit register, and S_{OUT} is the scan-out for the 8-bit register. Q_i ($i = 1, 2, \cdots, 8$) are the output values for the eight system latches. Z_i ($i = 1, 2, \cdots, 8$) are the inputs from the combinational logic. The structure that this network will be embedded into will be discussed shortly.

There are three primary modes of operation for this register, as well as one secondary mode of operation for this register. The first is shown in Fig. 19(b)—that is, with B_1 and B_2 equal to 11. This is a Basic System Operation mode, in which the Z_i values are loaded into the L_i, and the outputs are available on Q_i for system operation. This would be your normal register function.

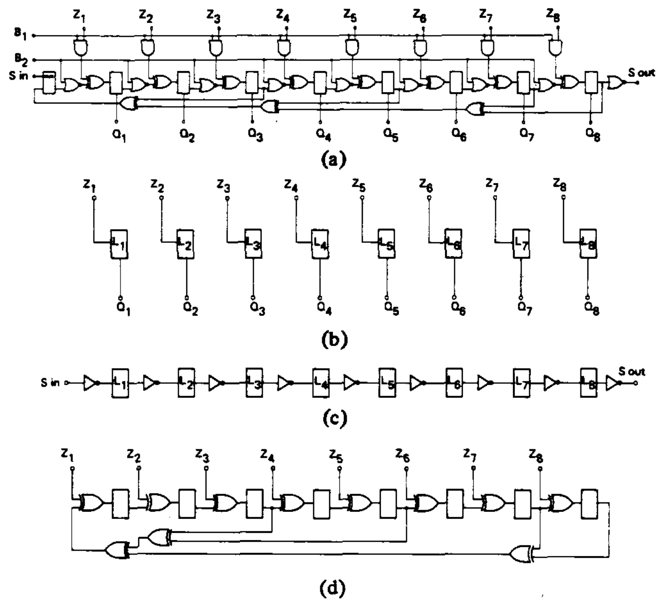


Fig. 19. BILBO and its different modes. (a) General form of BILBO register. (b) $B_1B_2 = 11$ system orientation mode. (c) $B_1B_2 = 00$ linear shift register mode. (d) $B_1B_2 = 10$ signature analysis register with m multiple inputs ($Z_1, Z_2, \cdots, Z_8$).

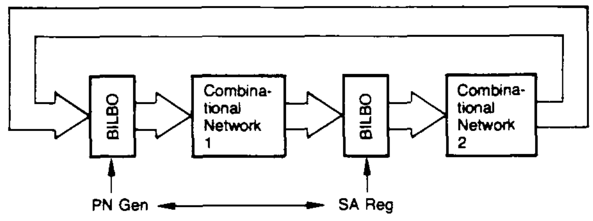


Fig. 20. Use of BILBO registers to test combinational Network 1.

When B_1B_2 equals 00, the BILBO register takes on the form of a linear shift register, as shown in Fig. 19(c). Scan-in input to the left, through some inverters, and basically lining up the eight registers into a single scan path, until the scan-out is reached. This is similar to Scan Path and LSSD.

The third mode is when B_1B_2 equals 10. In this mode, the BILBO register takes on the attributes of a linear feedback shift register of maximal length with multiple linear inputs. This is very similar to a Signature Analysis register, except that there is more than one input. In this situation, there are eight unique inputs. Thus after a certain number of shift clocks, say, 100, there would be a unique signature left in the BILBO register for the good machine. This good machine signature could be off-loaded from the register by changing from Mode $B_1B_2 = 10$ to Mode $B_1B_2 = 00$, in which case a shift register operation would exist, and the signature then could be observed from the scan-out primary output.

The fourth function that the BILBO register can perform is B_1B_2 equal to 01, which would force a reset on the register. (This is not depicted in Fig. 19.)

The BILBO registers are used in the system operation, as shown in Fig. 20. Basically, a BILBO register with combinational logic and another BILBO register with combinational logic, as well as the output of the second combinational logic network can feed back into the input of the first BILBO regis-

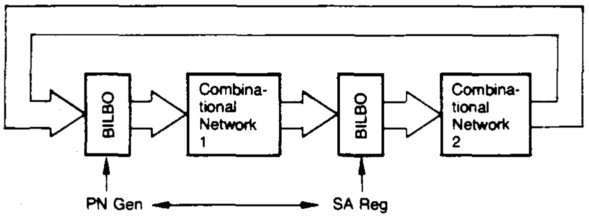


Fig. 21. Use of BILBO registers to test combinational Network 2.

ter. The BILBO approach takes one other fact into account, and that is that, in general, combinational logic is highly susceptible to random patterns. Thus if the inputs to the BILBO register, $Z_1, Z_2, \cdots, Z_8$, can be controlled to fixed values, such that the BILBO register is in the maximal length linear feedback shift register mode (Signature Analysis) it will output a sequence of patterns which are very close to random patterns. Thus random patterns can be generated quite readily from this register. These sequences are called Pseudo Random Patterns (PN).

If, in the first operation, this BILBO register on the left in Fig. 20 is used as the PN generator—that is, its data inputs are held to fixed values—then the output of that BILBO register will be random patterns. This will then do a reasonable test, if sufficient numbers of patterns are applied, of the Combinational Logic Network 1. The results of this test can be stored in a Signature Analysis register approach with multiple inputs to the BILBO register on the right. After a fixed number of patterns have been applied, the signature is scanned out of the BILBO register on the right for good machine compliance. If that is successfully completed, then the roles are reversed, and the BILBO register on the right will be used as a PN sequence generator; the BILBO register on the left will then be used as a Signature Analysis register with multiple inputs from Combinational Logic Network 2, see Fig. 21. In this mode, the Combinational Logic Network 2 will have random patterns applied to its inputs and its outputs stored in the BILBO register on the far left. Thus the testing of the combinational logic networks 1 and 2 can be completed at very high speeds by only applying the shift clocks, while the two BILBO registers are in the Signature Analysis mode. At the conclusion of the tests, off-loading of patterns can occur, and determination of good machine operation can be made.

This technique solves the problem of test generation and fault simulation if the combinational networks are susceptible to random patterns. There are some known networks which are not susceptible to random patterns. They are Programmable Logic Arrays (PLA's), see Fig. 22. The reason for this is that the fan-in in PLA's is too large. If an AND gate in the search array had 20 inputs, then each random pattern would have $1/2^{20}$ probability of coming up with the correct input pattern. On the other hand, random combinational logic networks with maximum fan-in of 4 can do quite well with random patterns.

The BILBO technique solves another problem and that is of test data volume. In LSSD, Scan Path, Scan/Set, or Random-Access Scan, a considerable amount of test data volume is involved with the shifting in and out. With BIBLO, if 100 patterns are run between scan-outs, the test data volume may be reduced by a factor of 100. The overhead for this technique is higher than for LSSD since about two EXCLUSIVE-OR's must be used per latch position. Also, there is more delay in the system data path (one or two gate delays). If VLSI has the huge number of logic gates available than this may be a very efficient way to use them.

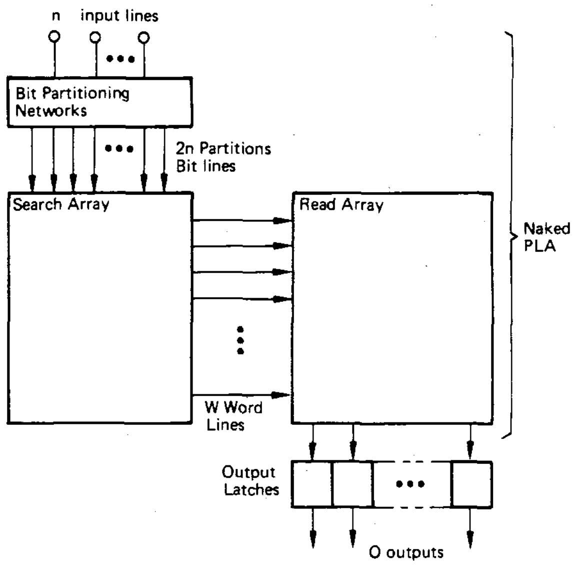


Fig. 22. PLA model.

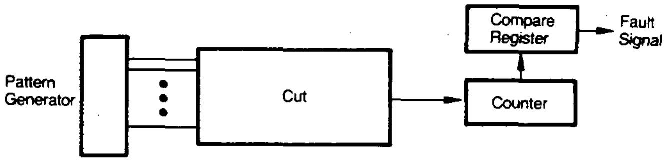


Fig. 23. Syndrome test structure.

B. Syndrome Testing

Recently, a technique was shown which could be used to test a network with fairly minor changes to the network. The technique is Syndrome Testing. The technique requires that all 2^n patterns be applied to the input of the network and then the number of 1's on the output be counted [115], [116].

Testing is done by comparing the number of 1's for the good machine to the number of 1's for the faulty machine. If there is a difference, the fault(s) in the faulty machine are detected (or Syndrome testable). To be more formal the Syndrome is:

Definition 1: The *Syndrome S* of a Boolean function is defined as

$$S = \frac{K}{2^n}$$

where K is the number of minterns realized by the function, and n is the number of binary input lines to the Boolean function.

Not all Boolean functions are totally Syndrome testable for all the single stuck-at-faults. Procedures are given in [115] with a minimal or near minimal number of primary inputs to make the networks Syndrome testable. In a number of "real networks" (i.e., SN74181, etc.) the numbers of extra primary inputs needed was at most one (<5 percent) and not more than two gates (<4 percent) were needed. An extension [116] to this work was published which showed a way of making a network Syndrome testable by adding extra inputs. This resulted in a somewhat longer test sequence. This is accomplished by holding some input constant while applying all 2^k inputs ($k < n$) then holding others constant and applying 2^l input patterns to l inputs. Whether the network is modified or not, the test data volume for a Syndrome testable design is extremely low. The general test setup is shown in Fig. 23.

The structure requires a pattern generator which applies all possible patterns once, a counter to count the 1's, and a com-

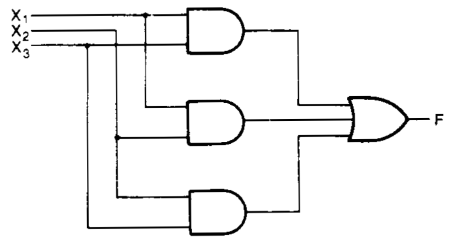


Fig. 24. Function to be tested with Walsh coefficients.

TABLE I
EXAMPLES OF WALSH FUNCTIONS AND WALSH COEFFICIENTS

$X_1X_2X_3$	W_2	$W_{1,3}$	F	W_2F	$W_{1,3}F$	W_{ALL}	$W_{ALL}F$
0 0 0	−1	+1	0	+1	−1	+1	−1
0 0 1	−1	−1	0	+1	+1	−1	−1
0 1 0	+1	+1	0	−1	−1	−1	−1
0 1 1	+1	−1	1	+1	−1	+1	−1
1 0 0	−1	−1	0	+1	+1	−1	−1
1 0 1	−1	+1	1	−1	+1	+1	−1
1 1 0	+1	−1	1	+1	−1	+1	−1
1 1 1	+1	+1	1	+1	+1	−1	+1
							$C_{ALL} = 4$

pare network. The overhead quoted is necessary to make the CUT Syndrome testable and does not include the pattern generator, counter, or compare register.

C. Testing by Verifying Walsh Coefficients

A technique which is similar to Syndrome Testing, in that it requires all possible input patterns be applied to the combinational network, is testing by verifying Walsh coefficients [117]. This technique only checks two of the Walsh coefficients and then makes conclusions about the network with respect to stuck-at-faults.

In order to calculate the Walsh coefficients, the logical value 0 (1) is associated with the arithmetic value −1(+1). There are 2^n Walsh functions. W_0 is defined to be 1, W_i is derived from all possible (arithmetic) products of the subject of independent input variables selected for that Walsh function. Table I shows the Walsh function for W_2, $W_{1,3}$, then W_2F, $W_{1,3}F$, finally W_{all} and $W_{\text{all}}F$. These values are calculated for the network in Fig. 24. If the values are summed for $W_{\text{all}}F$, the Walsh coefficient C_{all} is calculated. The Walsh coefficient C_0 is just W_0F summed. This is equivalent to the Syndrome in magnitude times 2^n. If $C_{\text{all}} \neq 0$ then all stuck-at-faults on primary inputs will be detected by measuring C_{all}. If the fault is present $C_{\text{all}} = 0$. If the network has $C_{\text{all}} = 0$ it can be easily modified such that $C_{\text{all}} \neq 0$. If the network has reconvergent fan-out then further checks need to be made (the number of inverters in each path has a certain property); see [117]. If these are successful, then by checking C_{all} and C_0, all the single stuck-at-faults can be detected. Some design constraints maybe needed to make sure that the network is testable by measuring C_{all} and C_0. Fig. 25 shows the network needed to determine C_{all} and C_0. The value p is the parity of the driving counter and the response counter is an up/down counter. Note, two passes must be made of the driving counter, one for C_{all} and one for C_0.

D. Autonomous Testing

The fourth technique which will be discussed in the area of self-test/built-in-test is Autonomous Testing [118]. Autonomous Testing like Syndrome Testing and testing Walsh coefficients requires all possible patterns be applied to the network inputs. However, with Autonomous Testing the outputs of

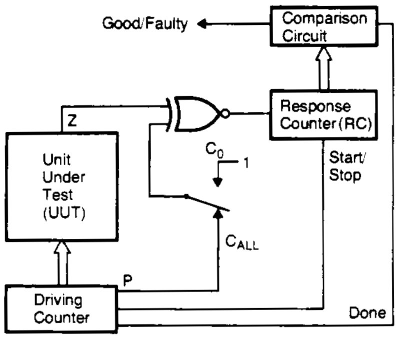


Fig. 25. Tester for veryfying C_0 and C_{all} Walsh coefficients.

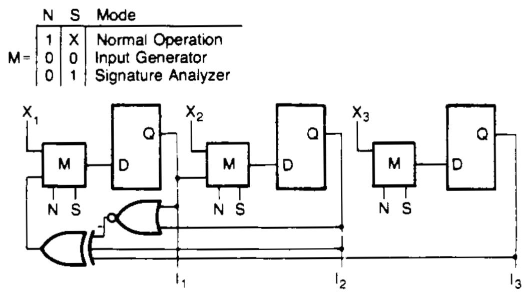


Fig. 26. Reconfigurable 3-bit LFSR module.

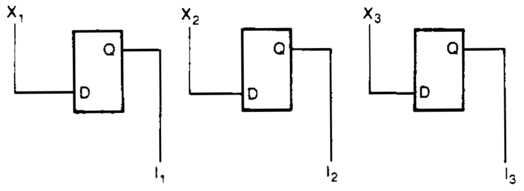


Fig. 27. Reconfigurable 3-bit LFSR module.

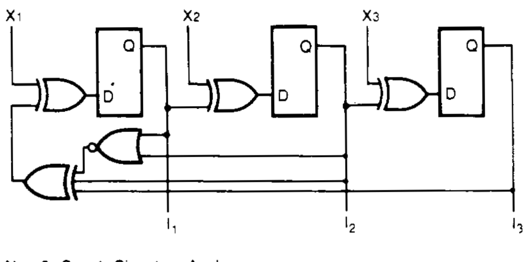
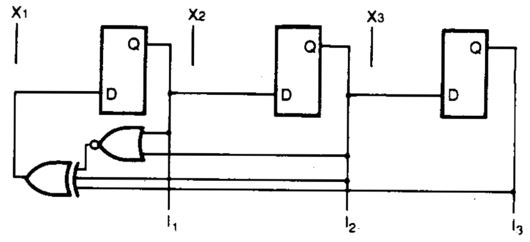


Fig. 28. Reconfigurable 3-bit LFSR module.

the network must be checked for each pattern against the value for the good machine. The results is that irrespective of the fault model Autonomous Testing will detect the faults (assuming the faulty machine does not turn into a sequential machine from a combinational machine). In order to help the network apply its own patterns and accumulate the results of the tests rather than observing every pattern for 2^n input patterns, a structure similar to BILBO register is used. This register has some unique attributes and is shown in Figs. 26-29. If a combinational network has 100 inputs, the network must be modified such that the subnetwork can be verified and, thus, the whole network will be tested.

Two approaches to partitioning are presented in the paper "Design for Autonomous Test" [118]. The first is to use



N = 0, S = 0: Input Generator

Fig. 29. Reconfigurable 3-bit LFSR module.

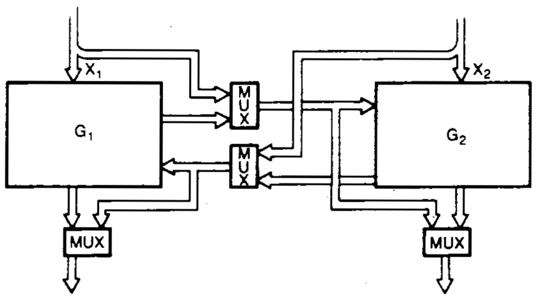


Fig. 30. Autonomous Testing—general network.

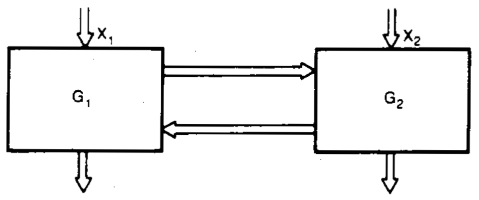


Fig. 31. Autonomous Testing—functional mode.

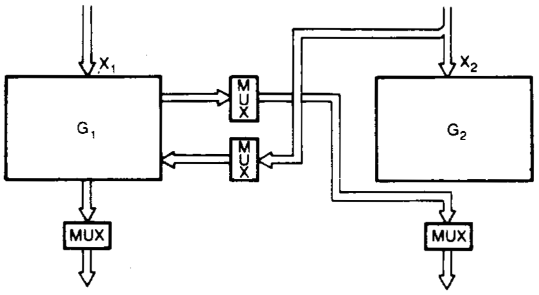


Fig. 32. Autonomous Testing—configuration to test network G_1.

multiplexers to separate the network and the second is a Sensitized Partitioning to separate the network. Fig. 30 shows the general network with multiplexers, Fig. 31 shows the network in functional mode, and Fig. 32 shows the network in a mode to test subnetwork G_1. This approach could involve a significant gate overhead to implement in some networks. Thus the Sensitized Partitioning approach is put forth. For example, the 74181 ALU/Function Generator is partitioned using the Sensitized Partitioning. By inspecting the network, two types of subnetworks can be partitioned out, four subnetworks N_1, one subnetwork N_2 (Figs. 33 and 34). By further inspection, all the L_i outputs of network N_1 can be tested by holding $S_2 = S_3 =$ low. Further, all the H_i outputs of network N_1 can be tested by holding $S_0 = S_1 =$ high, since sensitized paths exist through the subnetwork N_2. Thus far fewer than 2^n input patterns can be applied to the network to test it.

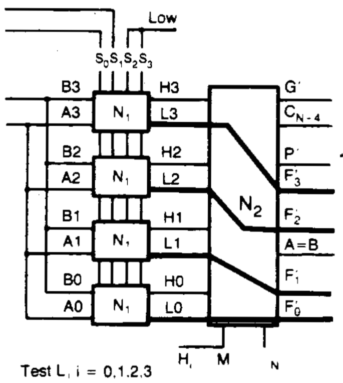


Fig. 33. Autonomous Testing with sensitized partitioning.

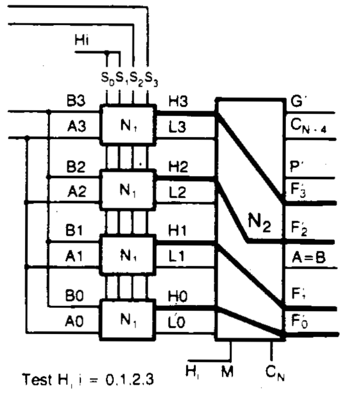


Fig. 34. Autonomous Testing with sensitized partitioning.

VI. Conclusion

The area of Design for Testability is becoming a popular topic by necessity. Those users of LSI/VLSI which do not have their own captive IC facilities are at the mercy of the vendors for information. And, until the vendor information is drastically changed, the Ad Hoc approaches to design for testability will be the only answer.

In that segment of the industry which can afford to implement the Structured Design for Testability approach, there is considerable hope of getting quality test patterns at a very modest cost. Furthermore, many innovative techniques are appearing in the Structured Approach and probably will continue as we meander through VLSI and into more dense technologies.

There is a new opportunity arriving in the form of gate arrays that allow low volume users access to VLSI technology. If they choose, structured design disciplines can be utilized. Perhaps "Silicon Foundries" of the future will offer a combined package of structured, testable modules and support software to automatically provide the user with finished parts AND tests.

Acknowledgment

The authors wish to thank D. J. Brown for his helpful comments and suggestions. The assistance of Ms. B. Fletcher, Ms. C. Mendoza, Ms. L. Clark, Ms. J. Allen, and J. Smith in preparing this manuscript for publication was invaluable.

References

General References and Surveys

[1] M. A. Breuer, Ed., *Diagnosis and Reliable Design of Digital Systems.* Rockville, MD: Computer Science Press, 1976.
[2] H. Y. Chang, E. G. Manning, and G. Metze, *Fault Diagnosis of*

Digital Systems. New York: Wiley-Interscience, 1970.
[3] A. D. Friedman and P. R. Menon, *Fault Detection in Digital Circuits.* Englewood Cliffs, NJ: Prentice-Hall, 1971.
[4] F. C. Hennie, *Finite State Models for Logical Machines.* New York: Wiley, 1968.
[5] P. G. Kovijanic, in "A new look at test generation and verification," in *Proc. 14th Design Automation Conf.*, IEEE Pub. 77CH1216-1C, pp. 58–63, June 1977.
[6] E. I. Muehldorf, "Designing LSI logic for testability," in *Dig. Papers, 1976 Ann. Semiconductor Test Symp.*, IEEE Pub. 76CH1179-1C, pp. 45–49, Oct. 1976.
[7] E. I. Muehldorf and A. D. Savkar, "LSI logic testing–An overview," *IEEE Trans. Comput.*, vol. C-30, no. 1, pp. 1–17, Jan. 1981.
[8] W. W. Peterson and E. J. Weldon, *Error Correcting Codes.* Cambridge, MA: MIT Press, 1972.
[9] A. K. Susskind, "Diagnostics for logic networks," *IEEE Spectrum*, vol. 10, pp. 40–47, Oct. 1973.
[10] T. W. Williams and K. P. Parker, "Testing logic networks and design for testability," *Computer*, pp. 9–21, Oct. 1979.
[11] IEEE, Inc., *IEEE Standard Dictionary of Electrical and Electronics Terms.* New York: Wiley-Interscience, 1972.

Designing for Testability

[12] "A designer's guide to signature analysis," Hewlett-Packard Application Note 222, Hewlett Packard, 5301 Stevens Creek Blvd., Santa Clara, CA 95050.
[13] S. B. Akers, "Partitioning for testability," *J. Des. Automat. Fault-Tolerant Comput.*, vol. 1, no. 2, Feb. 1977.
[14] H. Ando, "Testing VLSI with random access scan," in *Dig. Papers Compcon 80*, IEEE Pub. 80CH1491-OC, pp. 50–52, Feb. 1980.
[15] P. Bottorff and E. I. Muehldorf, "Impact of LSI on complex digital circuit board testing," *Electro 77*, New York, NY, Apr. 1977.
[16] S. DasGupta, E. B. Eichelberger, and T. W. Williams, "LSI chip design for testability," in *Dig. Tech. Papers, 1978 Int. Solid-State Circuits Conf.* (San Francisco, CA, Feb. 1978), pp. 216–217.
[17] "Designing digital circuits for testability," Hewlett-Packard Application Note 210-4, Hewlett Packard, Loveland, CO 80537.
[18] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," *J. Des. Automat. Fault-Tolerant Comput.*, vol. 2, no. 2, pp. 165–178, May 1978.
[19] —, "A logic design structure for LSI testing," in *Proc. 14th Design Automation Conf.*, IEEE Pub. 77CH1216-1C, pp. 462–468, June 1977.
[20] E. B. Eichelberger, E. J. Muehldorf, R. G. Walter, and T. W. Williams, "A logic design structure for testing internal arrays," in *Proc. 3rd USA-Japan Computer Conf.* (San Francisco, CA, Oct. 1978), pp. 266–272.
[21] S. Funatsu, N. Wakatsuki, and T. Arima, "Test generation systems in Japan," in *Proc. 12th Design Automation Symp.*, pp. 114–122, June 1975.
[22] H. C. Godoy, G. B. Franklin, and P. S. Bottoroff, "Automatic checking of logic design structure for compliance with testability groundrules," in *Proc. 14th Design Automation Conf.*, IEEE Pub. 77CH1216-1C, pp. 469–478, June 1977.
[23] J. P. Hayes, "On modifying logic networks to improve their diagnosability," *IEEE Trans. Comput.*, vol. C-23, pp. 56–62, Jan. 1974.
[24] J. P. Hayes and A. D. Friedman, "Test point placement to simplify fault detection," in *FTC-3, Dig. Papers, 1973 Symp. on Fault-Tolerant Computing*, pp. 73–78, June 1973.
[25] B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-in logic block observation techniques," in *Dig. Papers, 1979 Test Conf.*, IEEE Pub. 79CH1509-9C, pp. 37–41, Oct. 1979.
[26] M. D. Lippman and E. S. Donn, "Design forethought promotes easier testing of microcomputer boards," *Electronics*, pp. 113–119, Jan. 18, 1979.
[27] H. J. Nadig, "Signature analysis-concepts, examples, and guidelines," *Hewlett-Packard J.*, pp. 15–21, May 1977.
[28] M. Neil and R. Goodner, "Designing a serviceman's needs into microprocessor based systems," *Electronics*, pp. 122–128, Mar. 1, 1979.
[29] S. M. Reddy, "Easily testable realization for logic functions," *IEETC Trans. Comput.*, vol. C-21, pp. 1183–1188, Nov. 1972.
[30] K. K. Saliya and S. M. Reddy, "On minimally testable logic networks," *IEEE Trans. Comput.*, vol. C-23, pp. 1204–1207, Nov. 1974.
[31] J. H. Stewart, "Future testing of large LSI circuit cards," in *Dig. Papers 1977 Semiconductor Test Symp.*, IEEE Pub. 77CH1261-7C, pp. 6–17, Oct. 1977.
[32] A. Toth and C. Holt, "Automated data base-driven digital testing," *Computer*, pp. 13–19, Jan. 1974.
[33] E. White, "Signature analysis, enhancing the serviceability of microprocessor-based industrial products," in *Proc. 4th IECI Annual Conf.*, IEEE Pub. 78CH1312-8, pp. 68–76, Mar. 1978.
[34] M.J.Y. Williams and J. B. Angell, "Enhancing testability of large scale integrated circuits via test points and additional logic," *IEEE Trans. Comput.*, vol. C-22, pp. 46–60, Jan. 1973.
[35] T. W. Williams, "Utilization of a structured design for reliability and serviceability," in *Dig., Government Microcircuits Applications Conf.* (Monterey, CA, Nov. 1978), pp. 441–444.

Faults and Fault Modeling

[36] R. Boute and E. J. McCluskey, "Fault equivalence in sequential machines," in *Proc. Symp. on Computers and Automata* (Polytech. Inst. Brooklyn, Apr. 13–15, 1971), pp. 483–507.
[37] R. T. Boute, "Optimal and near-optimal checking experiments for output faults in sequential machines," *IEEE Trans. Comput.*, vol. C-23, no. 11, pp. 1207–1213, Nov. 1974.
[38] —, "Equivalence and dominance relations between output faults in sequential machines," Tech. Rep. 38, SU-SEL-72-052, Stanford Univ., Stanford, CA, Nov. 1972.
[39] F.J.O. Dias, "Fault masking in combinational logic circuits," *IEEE Trans. Comput.*, vol. C-24, pp. 476–482, May 1975.
[40] J. P. Hayes, "A NAND model for fault diagnosis in combinational logic networks," *IEEE Trans. Comput.*, vol. C-20, pp. 1496–1506, Dec. 1971.
[41] E. J. McCluskey and F. W. Clegg, "Fault equivalence in combinational logic networks," *IEEE Trans. Comput.*, vol. C-20, pp. 1286–1293, Nov. 1971.
[42] K.C.Y. Mei, "Fault dominance in combinational circuits," Tech. Note 2, Digital Systems Lab., Stanford Univ., Aug. 1970.
[43] —, "Bridging and stuck-at faults," *IEEE Trans. Comput.*, vol. C-23, no. 7, pp. 720–727, July 1974.
[44] R. C. Ogus, "The probability of a correct output from a combinational circuit," *IEEE Trans. Comput.*, vol. C-24, no. 5, pp. 534–544, May 1975.
[45] K. P. Parker and E. J. McCluskey, "Analysis of logic circuits with faults using input signal probabilities," *IEEE Trans. Comput.*, vol. C-24, no. 5, pp. 573–578, May 1975.
[46] K. K. Saliya and S. M. Reddy, "Fault detecting test sets for Reed-Muller canonic networks," *IEEE Trans. Comput.*, pp. 995–998, Oct. 1975.
[47] D. R. Schertz and G. Metze, "A new representation for faults in combinational digital circuits," *IEEE Trans. Comput.*, vol. C-21, no. 8, pp. 858–866, Aug. 1972.
[48] J. J. Shedletsky and E. J. McCluskey, "The error latency of a fault in a sequential digital circuit," *IEEE Trans. Comput.*, vol. C-25, no. 6, pp. 655–659, June 1976.
[49] —, "The error latency of a fault in a combinational digital circuit," in *FTCS-5, Dig. Papers, 5th Int. Symp. on Fault Tolerant Computing* (Paris, France, June 1975), pp. 210–214.
[50] K. To, "Fault folding for irredundant and redundant combinational circuits," *IEEE Trans. Comput.*, vol. C-22, no. 11, pp. 1008–1015, Nov. 1973.
[51] D. T. Wang, "Properties of faults and criticalities of values under tests for combinational networks," *IEEE Trans. Comput.*, vol. C-24, no. 7, pp. 746–750, July 1975.

Testing and Fault Location

[52] R. P. Batni and C. R. Kime, "A module level testing approach for combinational networks," *IEEE Trans. Comput.*, vol. C-25, no. 6, pp. 594–604, June 1976.
[53] S. Bisset, "Exhaustive testing of microprocessors and related devices: A practical solution," in *Dig. Papers, 1977 Semiconductor Test Symp.*, pp. 38–41, Oct. 1977.
[54] R. J. Czepiel, S. H. Foreman, and R. J. Prilik, "System for logic, parametric and analog testing," in *Dig. Papers, 1976 Semiconductor Test Symp.*, pp. 54–69, Oct. 1976.
[55] R. A. Frohwerk, "Signature analysis: A new digital field service method," *Hewlett-Packard J.*, pp. 2–8, May 1977.
[56] B. A. Grimmer, "Test techniques for circuit boards containing large memories and microprocessors," in *Dig. Papers, 1976 Semiconductor Test Symp.*, pp. 16–21, Oct. 1976.
[57] W. A. Groves, "Rapid digital fault isolation with FASTRACE," *Hewlett-Packard J.*, pp. 8–13, Mar. 1979.
[58] J. P. Hayes, "Rapid count testing for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-25, no. 6, pp. 613–620, June 1976.
[59] —, "Detection of pattern sensitive faults in random access memories," *IEEE Trans. Comput.*, vol. C-24, no. 2, Feb. 1975, pp. 150–160.
[60] —, "Testing logic circuits by transition counting," in *FTC-5, Dig. Papers, 5th Int. Symp. on Fault Tolerant Computing* (Paris, France, June 1975), pp. 215–219.
[61] J. T. Healy, "Economic realities of testing microprocessors," in *Dig. Papers, 1977 Semiconductor Test Symp.*, pp. 47–52, Oct. 1977.
[62] E. C. Lee, "A simple concept in microprocessor testing," in *Dig.*

Papers, 1976 Semiconductor Test Symp., IEEE Pub. 76CH1179-1C, pp. 13-15, Oct. 1976.

[63] J. Losq, "Referenceless random testing," in *FTCS-6, Dig. Papers, 6th Int. Symp. on Fault-Tolerant Computing* (Pittsburgh, PA, June 21-23, 1976), pp. 81-86.

[64] S. Palmquist and D. Chapman, "Expanding the boundaries of LSI testing with an advanced pattern controller," in *Dig. Papers, 1976 Semicondctor Test Symp.*, pp. 70-75, Oct. 1976.

[65] K. P. Parker, "Compact testing: Testing with compressed data," in *FTCS-6, Dig. Papers, 6th Int. Symp. on Fault-Tolerant Computing* (Pittsburgh, PA, June 21-23, 1976).

[66] J. J. Shedletsky, "A rationale for the random testing of combinational digital circuits," in *Dig. Papers, Compcon 75 Fall Meet.* (Washington, DC, Sept. 9-11, 1975), pp. 5-9.

[67] V. P. Strini, "Fault location in a semiconductor random access memory unit," *IEEE Trans. Comput.*, vol. C-27, no. 4, pp. 379-385, Apr. 1978.

[68] C. W. Weller, in "An engineering approach to IC test system maintenance," in *Dig. Papers, 1977 Semiconductor Test Symp.*, pp. 144-145, Oct. 1977.

Testability Measures

[69] W. J. Dejka, "Measure of testability in device and system design," in *Proc. 20th Midwest Symp. Circuits Syst.*, pp. 39-52, Aug. 1977.

[70] L. H. Goldstein, "Controllability/observability analysis of digital circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-26, no. 9, pp. 685-693, Sept. 1979.

[71] W. L. Keiner and R. P. West, "Testability measures," presented at AUTOTESTCON '77, Nov. 1977.

[72] P. G. Kovijanic, "testability analysis," in *Dig. Papers, 1979 Test Conf.*, IEEE Pub. 79CH1509-9C, pp. 310-316, Oct. 1979.

[73] J. E. Stephenson and J. Grason, "A testability measure for register transfer level digital circuits," in *Proc. 6th Fault Tolerant Computing Symp.*, pp. 101-107, June 1976.

Test Generation

[74] V. Agrawal and P. Agrawal, "An automatic test generation system for ILLIAC IV logic boards," *IEEE Trans. Comput.*, vol. C-C-21, no. 9, pp. 1015-1017, Sept. 1972.

[75] D. B. Armstrong, "On finding a nearly minimal set of fault detection tests for combinational logic nets," *IEEE Trans. Electron. Comput.*, vol. EC-15, no. 1, pp. 66-73, Feb. 1966.

[76] R. Betancourt, "Derivation of minimum test sets for unate logical circuits," *IEEE Trans. Comput.*, vol. C-20, no. 11, pp. 1264-1269, Nov. 1973.

[77] D. C. Bossen and S. J. Hong, "Cause and effect analysis for multiple fault detection in combinational networks," *IEEE Trans. Comput.*, vol. C-20, no. 11, pp. 1252-1257, Nov. 1971.

[78] P. S. Bottorff *et al.*, "Test generation for large networks," in *Proc. 14th Design Automation Conf.*, IEEE Pub. 77CH1216-1C, pp. 479-485, June 1977.

[79] R. D. Edlred, "Test routines based on symbolic logic statements," *J. Assoc. Comput. Mach.*, vol. 6, no. 1, pp. 33-36, 1959.

[80] P. Goel, "Test generation costs analysis and projections," presented at the 17th Design Automation Conf., Minneapolis, MN, 1980.

[81] E. P. Hsieh *et al.*, "Delay test generation," in *Proc. 14th Design Automation Conf.*, IEEE Pub. 77CH1216-1C, pp. 486-491, June 1977.

[82] C. T. Ku and G. M. Masson, "The Boolean difference and multiple fault analysis," *IEEE Trans. Comput.*, vol. C-24, no. 7, pp. 691-695, July 1975.

[83] E. I. Muehldorf, "Test pattern generation as a part of the total design process," in *LSI and Boards: Dig. Papers, 1978 Ann. Semiconductor Test Symp.*, pp. 4-7, Oct. 1978.

[84] E. I. Muehldorf and T. W. Williams, "Optimized stuck fault test patterns for PLA macros," in *Dig. Papers, 1977 Semiconductor Test Symp.*, IEEE Pub. 77CH1216-7C, pp. 89-101, Oct. 1977.

[85] M. R. Page, "Generation of diagnostic tests using prime implicants," Coordinated Science Lab. Rep. R-414, University of Illinois, Urbana, May 1969.

[86] S. G. Papaioannou, "Optimal test generation in combinational networks by pseudo Boolean programming," *IEEE Trans. Comput.*, vol. C-26, no. 6, pp. 553-560, June 1977.

[87] K. P. Parker, "Adaptive random test generation," *J. Des. Automat. Fault Tolerant Comput.*, vol. 1, no. 1, pp. 62-83, Oct. 1976.

[88] ——, "Probabilistic test generation," Tech. Note 18, Digital Systems Laboratory, Stanford University, Stanford, CA, Jan. 1973.

[89] J. F. Poage and E. J. McCluskey, "Derivation of optimum tests for sequential machines," in *Proc. 5th Ann. Symp. on Switching Circuit Theory and Logic Design*, pp. 95-110, 1964.

[90] ——, "Derivation of optimum tests to detect faults in combinational circuits," in *Mathematical Theory of Automation.* New York: Polytechnic Press, 1963.

[91] G. R. Putzolu and J. P. Roth, "A heuristic algorithm for testing of asynchronous circuits," *IEEE Trans. Comput.*, vol. C-20, no. 6, pp. 639-647, June 1971.

[92] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 567-580, Oct. 1967.

[93] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Devel.*, no. 10, pp. 278-281, Oct. 1966.

[94] P. R. Schneider, "On the necessity to examine D-chairs in diagnostic test generation—An example," *IBM J. Res. Develop.*, no. 11, p. 114, Nov. 1967.

[95] H. D. Schnurmann, E. Lindbloom, R. G. Carpenter, "The weighted random test pattern generation," *IEEE Trans. Comput.*, vol. C-24, no. 7, pp. 695-700, July 1975.

[96] E. F. Sellers, M. Y. Hsiao, and L. W. Bearnson, "Analyzing errors with the Boolean difference," *IEEE Trans. Comput.*, vol. C-17, no. 7, pp. 676-683, July 1968.

[97] D. T. Wang, "An algorithm for the detection of tests sets for combinational logic networks," *IEEE Trans. Comput.*, vol. C-25, no. 7, pp. 742-746, July 1975.

[98] T. W. Williams and E. E. Eichelberger, "Random patterns within a structured sequential logic design," in *Dig. Papers, 1977 Semiconductor Test Symp.*, IEEE Pub. 77CH1261-7C, pp. 19-27, Oct. 1977.

[99] S. S. Yau and S. C. Yang, "Multiple fault detection for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-24, no. 5, pp. 233-242, May 1975.

Simulation

[100] D. B. Armstrong, "A deductive method for simulating faults in logic circuits," *IEEE Trans. Comput.*, vol. C-22, no. 5, pp. 464-471, May 1972.

[101] M. A. Breuer, "Functional partitioning and simulation of digital circuits," *IEEE Trans. Comput.*, vol. C-19, no. 11, pp. 1038-1046, Nov. 1970.

[102] H.Y.P. Chiang *et al.*, "Comparison of parallel and deductive fault simulation," *IEEE Trans. Comput.*, vol. C-23, no. 11, pp. 1132-1138, Nov. 1974.

[103] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM J. Res. Devel.*, Mar. 1965.

[104] E. Manning and H. Y. Chang, "Functional technique for efficient digital fault simulation," in *IEEE Int. Conv. Dig.*, p. 194, 1968.

[105] K. P. Parker, "Software simulator speeds digital board test generation," *Hewlett-Packard J.*, pp. 13-19, Mar. 1979.

[106] S. Seshu, "On an improved diagnosis program," *IEEE Trans. Electron. Comput.*, vol. EC-12, no. 1, pp. 76-79, Feb. 1965.

[107] S. Seshu and D. N. Freeman, "The diagnosis of asynchronous sequential switching systems," *IRE Trans. Electron. Compat.*, vol. EC-11, no. 8, pp. 459-465, Aug. 1962.

[108] T. M. Storey and J. W. Barry, "Delay test simulation," in *Proc. 14th Design Automation Conf.*, IEEE Pub. 77CH1216-1C, pp. 491-494, June 1977.

[109] S. A. Szygenda and E. W. Thompson, "Modeling and digital simulation for design verification diagnosis," *IEEE Trans. Comput.*, vol. C-25, no. 12, pp. 1242-1253, Dec. 1976.

[110] S. A. Szygenda, "TEGAS2—Anatomy of a general purpose test generation and simulation system for digital logic," in *Proc. 9th Design Automation Workshop*, pp. 116-127, 1972.

[111] S. A. Szygenda, D. M. Rouse, and E. W. Thompson, "A model for implementation of a universal time delay simulation for large digital networks," in *AFIPS Conf. Proc.*, vol. 36, pp. 207-216, 1970.

[112] E. G. Ulrich and T. Baker, "Concurrent simulation of nearly identical digital networks," *Computer*, vol. 7, no. 4, pp. 39-44, Apr. 1974.

[113] ——, "The concurrent simulation of nearly identical digital networks," in *Proc. 10th Design Automation Workshop*, pp. 145-150, June 1973.

[114] E. G. Ulrich, T. Baker, and L. R. Williams, "Fault test analysis techniques based on simulation," in *Proc. 9th Design Automation Workshop*, pp. 111-115, 1972.

[115] J. Savir, "Syndrome—Testable design of combinational circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 442-451, June 1980 (corrections: Nov. 1980).

[116] ——, "Syndrome—Testing of 'syndrome-untestable' combinational circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 606-608, Aug. 1981.

[117] A. K. Susskind, "Testing by verifying Walsh coefficients," in *Proc. 11th Ann. Symp. on Fault-Tolerant Computing* (Portland, MA), pp. 206-208, June 1981.

[118] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Trans. Comput.*, vol. C-30, pp. 866-875, Nov. 1981.