AUTOMATIC TESTING

This article describes the topic of automatic testing of electronic integrated circuits. Due to the variety of implementations and technologies, automatic testing takes several forms. Automatic testing first appeared in the form of automatic test equipment (ATE) to test newly manufactured integrated circuits. The automated operation of these devices facilitated the mass production of circuits. In ATE, the testing process consists of presenting a series of inputs to the circuits. Simultaneous with the presentation of inputs circuit outputs are compared against acceptable responses. In the event discrepancies appear, the devices were either scrapped or reconfigured into working circuits if the designs allowed. With the ever-increasing complexity of circuits, it was observed that some of the ATE capabilities could be integrated onto the same circuit to enable the circuit to either fully or partially test itself. In addition, built-in self-test circuitry is being integrated in increasingly complex chips conferring significant benefits in these applications.

The science of automatic testing includes topics from several areas of knowledge. First, the defects must be classified according to models that accurately render the defect behavior of the circuit, giving test engineers a target for developing defect tests. Second, given a defect model, methodologies for determining test patterns must be examined. Indeed, circuits can be inherently easy to develop tests for, highly testable, while others may require special design practices to become easily testable. Built-in test circuitry facilitates the testing of a circuit. By adding special test structures, the circuit can be more easily testable. Third, in many cases, the number of tests to give acceptable test performance may be large, especially for circuits that are not designed to be testable. Fourth, simulation can assist the test engineer in assessing the efficacy of a particular test regimen. Fault simulations are used to determine what faults can be detected with the test regimen providing valuable feedback in the test design process. Two excellent references on topics related to automatic testing are (1,2). These works provide comprehensive treatment of testing methodologies.

The ever-increasing complexity of integrated circuits and digital systems makes verifying that the circuit or system is fully functional more difficult. Circuit testers can control and observe system inputs and outputs, respectively, which may number in the thousands. The number of potential component failures in current systems can be in the range of billions. This mismatch between the number of inputs and outputs and the number of internal structures suggests that there are many challenges in testing these devices. If the direct effects of the faults must be seen at the circuit outputs, it is likely that an enormous number of test patterns must be presented to the circuit, unless attention is paid to efficient generation of test patterns, design for test methodologies, and/or built-in self-test methodologies are employed. By allowing internal structures to automatically test themselves, the internal structures need only report that they are fault-free or faulty.

Many vendors today supply modules and subsystems that can be included in larger designs. Because the design is considered intellectual property, many vendors do not want to release sufficient implementation details to enable construction of an effective test strategy. By incorporating Built-In Self-Test (BIST) capabilities in the subsystem, the vendor can supply a testable design without disclosing any details of the design. Importantly, the customer can achieve more acceptable testing results without needing to know the details of the design.

Not surprisingly, economics plays a role in the manner in which testing approaches are applied. One study has shown (3) that in consumer electronics, BIST approaches may not be cost-effective. Testing affects the profitability of a design in several ways. BIST requires the addition of circuitry that is not necessary to maintain functionality. This additional circuitry increases the size of the chip. This increase in chip size reduces the number of chips that can be manufactured per wafer and also reduces the chip yields as a consequence of the increased die size. For high volume products, the costs can be enormous. For example, Intel calculated in 1995 for the Pentium processor, a 1% increase in chip area resulted in a $63 million increase in production costs. A 15% increase in chip area resulted in an almost $1 billion increase in production costs. A study done in Ref. 3 showed that high production chips that are part of consumer electronics typically have a useful design life of 2 years. For these chips, the addition of automatic test capabilities increased costs when all cost factors were taken into account. On the other hand, designs with longer useful design lives, say 5 years, benefit from an automatic test capability.

This article is organized into sections including an introduction, a description of foundational testing paradigms, a description of commonly used automatic testing methodologies, CAD tool support, frontiers in automatic testing, and automatic testing case studies. In the section entitled "Testing Principles," the general principles and methods for testing are introduced. This topic is covered in more detail in other encyclopedia articles. In "Economics of Test," design for test methodologies are presented as they relate to automatic testing and in "Programmable Logic Arrays," the subject of test pattern generation is discussed. In the section entitled "Built-In Self-Test," these methodologies are discussed and in "CAD Tools," the CAD tool support available in current tools is presented. In "Analog BIST," the principles of automatic testing for analog systems are presented. In "Automatic Testing Case Studies," several actual implementations that employ automatic testing are presented, and in the next section the frontiers of automatic testing are described.

TESTING PRINCIPLES

In this section, several topics related to testing methodologies, techniques, and philosophy are discussed. Many auto-

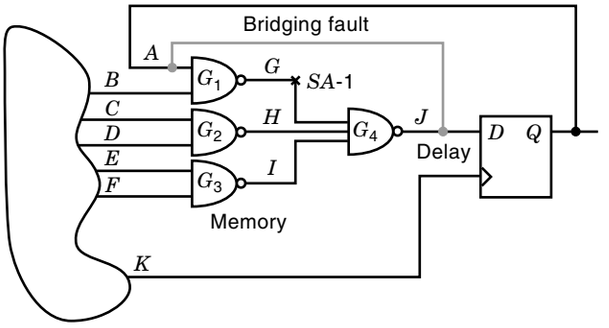

Figure 1. An illustration of fault models. Digital circuits are susceptible to many types of faults.

mated testing approaches are derived from the less restrictive testing methodologies. For a more in-depth discussion of testing techniques, the interested reader should see Refs. 1 and 2.

Fault Modeling

Circuits can fail in many ways. The failures can be a result of manufacturing defects, infant mortality, random failures, age, or external disturbances (4). The defects can be localized, affecting the function of one circuit element or distributed, affecting many or all circuit elements. The failures can result in temporary or permanent failure of the circuit. The quality and detail of the fault models can have an impact on the success of the test strategy. In addition, the fault model may have an impact on the overall test strategy. In order to develop effective testing methodologies, accurate models for circuit failures must be agreed upon and targeted by the testing approach. The fault models selected depend on the technology used to implement the circuits. Manufacturing defects exist as a consequence of the manufacture of the circuit. The introduction and study of manufacturing defects is a heavily studied topic because of its impact on the profitability of the device. Dust or other aerosols in the air can affect the defect statistics of a particular manufacturing run. In addition, mask misalignment and defects in the mask can also increase the defect densities. Figure 1 gives some example faults which are discussed in more detail in the following sections.

Stuck-at Fault Models. Stuck-at fault models are the simplest and most widely used fault models in testing. The stuck-at fault model requires the adoption of several fundamental assumptions. First, a stuck-at fault manifests itself as a node being stuck at either of the allowable logic levels, either zero or one, regardless of the inputs that are applied to the gate that drives the node. Second, the stuck-at fault model assumes that the faults are permanent. Third, the stuck-at fault model assumes that gates maintain their ordinary function in the presence of the fault. Significantly, the stuck-at fault model also models a common failure mode in digital circuits. The circuit shown in Fig. 1 can be used to illustrate the fault model. The output of gate G_1 can be stuck-at 1 as a result of a defect. When the fault is present, the corresponding input to G_4 will always be one. In order for the fault to manifest itself, a discrepancy must occur in the circuit as a consequence of the fault. To force the discrepancy, the circuit inputs are manipulated so that $A = B = 1$, with the discrepancy appearing at the output of the gate. This discrepancy may ultimately result in the system malfunction. A second example circuit is shown in Fig. 2, consisting of an OR gate (G_1) driving one input in three AND gates (G_2, G_3, and G_4). Consider the occurrence of a stuck-at 1 fault at the input to G_1, the fault results in the output being 1 as a consequence of the fault. In this simple example, one can thus observe the indistinguishability between output and input stuck at 1 faults. For modeling purposes, these faults can be collapsed into a single fault. In the event gate input I_2 has a stuck-at 0 or 1 fault, the situation is somewhat different. In this case, $O_1 = I_3 = I_4$ and G_3 and G_4 will not be directly affected by the fault.

Delay Fault Models. A delay fault is a fault where a part of the circuit operates more slowly, relative to other circuit structures when a fault is present. When such a fault is present, the circuit may operate correctly when operated at slower clock rates, but does not operate at speed under certain circumstances. Delay faults can be modeled at several levels (5). Gate delay fault models are modeled as excessive delay in a gate as a consequence of faults. The transition fault model is either a slow to transition from 0 to 1 or from 1 to 0. A path delay fault is present when the propagation delay through a series of gates is larger than some desired delay. Indeed, a current industry practice is to perform statistical timing analysis of parts. The manufacturer can determine that the parts can be run at a higher speed with a certain probability so that higher levels of performance can be delivered to customers. However, this relies on the statistical likelihood that delays will not be worst case (5). By running the device at a higher clock rate, devices and structures that satisfy worst-case timing along the critical path, may not meet the timing at the new higher clock rate. Hence, a delay fault can appear as a consequence of the manufacturing decisions. Assuming the

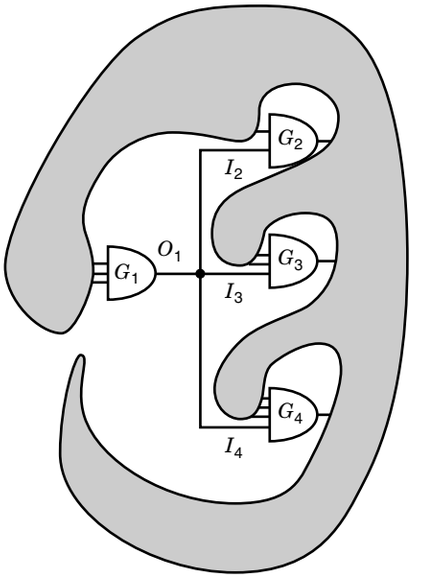

Figure 2. Illustration of input stuck-at faults. O_1 stuck-at O forces the outputs of gates G_2, G_3, and G_4 to be stuck-at 0.

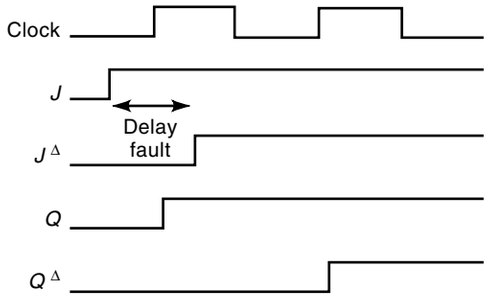

Figure 3. Illustration of a delay fault. The presence of a delay fault causes the wrong value to be clocked into the flip-flop.

indicated delay fault in Fig. 1, Fig. 3 gives a timing diagram showing manifestation of the fault. In this circuit, the delay fault causes the flip-flop input, J, to be delayed for a particular combination of inputs and input change(s), resulting in value being stored in the flip-flop being delayed by one clock period. Because of the nature of the delay fault, the circuit must be tested at speed in order to detect the delay fault because at slower clocks, the circuit will operate correctly since the circuit functions correctly at slower clocks. Furthermore, because the delay fault is dynamic, in order to detect the fault, the combinational circuit must receive an input change and the flip-flop must be clocked to make the delay fault observable.

Bridging Faults. Bridging faults are the presence of an undesirable electrical connection between two nodes. This connection results in the circuit malfunctioning or behaving in a degraded fashion. Furthermore, bridging faults may manifest themselves in wired-and or wired-or fashions, changing the circuit function. In addition to degrading the signal, the bridging fault may be manifested as stuck faults when bridging faults occur between a node and the supply lines or as a sequential circuit when the bridging fault creates a feedback connection (5a). Bridging faults require physical proximity between the circuit structures afflicted by the bridging faults. Figure 1 gives an example of a bridging fault that changes the combinational circuit into a sequential circuit.

CMOS Fault Models. CMOS technology has several fault modes that are unique to the technology (5a). Furthermore, as a consequence of the properties of the technology, CMOS offers alternative methods for identifying defective circuits. CMOS gates consist of complementary networks of PMOS and NMOS transistors configured such that significant currents may be drawn only when signal changes occur. When no signal changes occur, the normally working circuit draws very low leakage currents.

Since the CMOS circuit should only draw currents when the circuit is switching, any significant divergence from a known current profile is indicative of faults. For example, the gates may logically function correctly, but as a consequence of the faults being present, the circuit may show abnormally large power supply currents. These changes in current draw characteristics can be used as a diagnostic for indicating the presence of faults and in possibly identifying the faults. Testing for faults based on this observation is called I_{DDQ} testing. Bridging faults are common in CMOS circuits (6) and are effectively detected with I_{DDQ} testing (7). I_{DDQ} faults can have a significant impact on portable designs where the low current drawn by CMOS circuits is necessary.

CMOS circuits also have an interesting failure mode where an ordinary gate can be turned into a sequential circuit. The fault is a consequence of a transistor failure, low quiescent currents, and capacitive gate inputs. In Fig. 4, if transistor Q_1 is stuck open, the gate input to the inverter, G_1 dynamically stores the prior value on node C when $A = 0$ and $B = 1$. In order to make this fault visible, C must be forced to 1 by setting $A = B = 0$ followed by setting $B = 1$ to store the value at the input of G_2.

Memory Faults. Semiconductor memories have structures that are very regular and very dense. As a result, they can exhibit faults that are not ordinarily seen in other circuits which can complicate the testing process. The faults can affect the memory behavior in unusual ways (8). First, a fault can link two memory cells in such a way that when a value is written into one cell, the value toggles in another cell. Second, the memory cell can only be written to 0 or 1 but cannot be written the opposite value. Third, the behavior of a memory cell may be sensitive to the contents of the neighboring cells. For example, a particular pattern values stored in surrounding cells may prevent writing into the affected cell. Fourth, the particular pattern of values stored in the cells can result in the value in the affected cell changing. The nature of these faults make their detection challenging.

Crosspoint Faults. Crosspoint faults (1) are a type of defect that can occur in PLAs (Programmable Logic Arrays). PLAs consist of AND arrays and OR arrays with individual terms included in each through the programming of transistors that either include or exclude a term through the presence or absence of connections in the array. In field programmable devices, a transistor is programmed to be on or off, respectively, to represent the presence or absence of a connection. A crosspoint fault is the undesired presence or absence of a connection in the PLA. The crosspoint fault can result in a change in the logic function that cannot be modeled by the stuck fault model. A crosspoint fault with a missing connection in the AND array results in a product term of fewer variables while an extra connection results in more variables in the product term. For example, consider function $f(A, B, C, D) = AB + CD$ implemented on a PLA. The existence of a crosspoint fault can change the function to $f^{cpf}(A, B, C, D) = ABC + CD$. Fig-

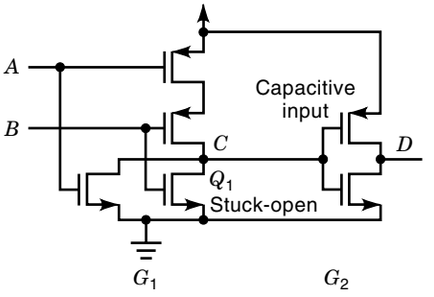

Figure 4. An illustration of a CMOS memory fault. CMOS circuits can suffer faults that impart sequential behaviors.

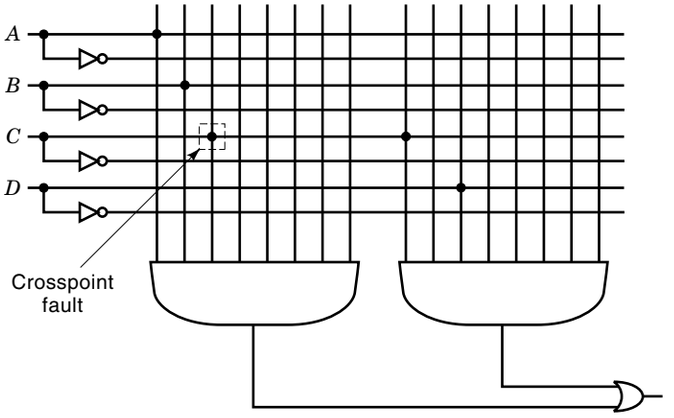

Figure 5. An illustration of a crosspoint fault. The crosspoint fault results in the programmable logic array to evaluate the wrong function.

ure 5 diagrams the structure of the PLA and the functional effect of the crosspoint fault.

Measures of Testing

In order to gauge the success of a test methodology, some measure of the testing success and overheads are necessary. In this section, the measures of test coverage, test set size, hardware overhead, and performance impact are discussed.

Test Coverage. Test coverage is the percentage of targeted faults that have been covered by the test regimen. Ideally, 100% test coverage is desired; however, this can be misleading if the fault model does not accurately reflect the types of faults that can be expected to occur (8a). For example, the stuck fault model is a popular and simple model for faults that works well in many situations. CMOS circuits, however, have several failure modes that cannot be modeled by the stuck fault model. A stuck fault test can be constructed that covers 100% of the stuck faults, yet may be only partially successful in identifying other faults. Fault coverage is determined through fault simulation of the respective circuit. In order to assess the performance of a test, a fault simulator must be able to accurately model the targeted fault to get a realistic measure of fault coverage.

Size of Test Set. The size of the test set is an indirect measure of the complexity of the test set. The size of the test set impacts the test time which has a direct impact on circuit cost if expensive circuit testers are employed. In addition, the test set size is related to the effort, both in personnel and computationally, that was required to develop the test. The size of the test set depends on many factors including the ease with which the design can be tested as well as implementation of Design for Test (DFT) methodologies. Indeed, DFT methodologies may not necessarily result in shorter tests. Use of scan path approaches where flip-flops are interconnected as shift registers gives excellent fault coverages, yet the process of scanning into and out from the shift register can result in large test sets.

Hardware Overhead. The addition of circuitry to improve testability or incorporate BIST capabilities can increase the size of a chip or system and as shown previously can have a disproportionate impact on circuit costs. In the event improved testability is a requirement, the increase in hardware can be used as a criteria for evaluating different designs. The ratio of the circuit size with test circuitry to the circuit without test circuitry is a straightforward measure of the hardware overhead. The additional circuitry for test can increase the likelihood that defects are present in the test circuitry. In addition, failure rates of circuits in service are a function of the size of the circuit, where larger circuits have higher failure rates.

Impact on Performance. Likewise, the addition of test circuitry can have an impact on system performance. The impact can be measured in terms of reduced clock rate and higher power requirements. For example, scan design methods add circuitry to flip-flops that multiplex between normal and test modes which typically have larger delays as compared to circuits not equipped. For devices with fixed die sizes and PLAs, the addition of test circuitry may be at the expense of circuitry that improves the performance of operation.

Testability. Testability is an analysis and metric that describes how easily a design may be tested for defects. In testing a circuit for defects, the goal is to supply inputs to the circuit so that it behaves correctly when no defects are present, but which malfunctions if a single defect is present. In other words, the only way to detect the defect is to force the circuit to malfunction. In general, testability is measured in terms of the specific and collective observability and controllability of nodes within a design. For example, a circuit which gives the test engineer direct access (setting and reading) to flip-flop contents is more easily testable than one does not, which would give a corresponding better testability figure. In the test community, testability is often described in the context of controllability and observability. Controllability of a circuit node is the capability of being able to set the node to a particular value. Observability of a circuit node is the ability of being able to observe the value of the node (either complemented or uncomplemented) at the circuit outputs. Estimates of the difficulties of controlling and observing circuit nodes form the basis for testability measures. Figure 6 presents a simple illustration of the problem and process. The node S is susceptible to many types of faults. The general procedure for testing for the correct operation of node S is to control the node to a value contrary to the fault value. Next, the observed value of the signal is communicated to the outputs of the system for observation. Detecting faults with redundancy of any sort requires special consideration in order to be able to detect all possible faults. For example, fault tol-

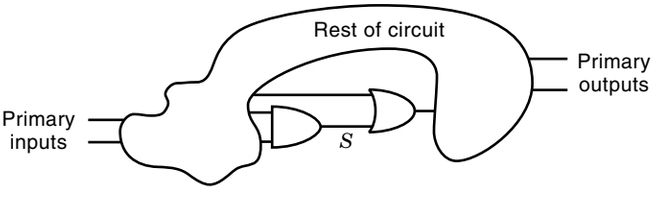

Figure 6. Representative circuit with fault. In order to test for faults on S, the node must be controlled by the inputs and observed at the outputs.

erant systems that employ triple modular redundancy (TMR) will not show any discrepancies at the circuit outputs when one fault is present (4). In order to make the modules testable, they must be somehow separated so that the redundancy does not mask the presence of faults. In addition, redundant gates necessary to remove hazards from combinational circuits result in a circuit where certain faults are untestable. Testability can be achieved by making certain internal nodes observable.

Automatic Test Equipment

Automatic test equipment tests integrated circuits by applying a set of inputs and comparing output responses with known good responses. The equipment can be used to determine whether a circuit works or not. If desired, the tests can be constructed to indicate what part of the circuit has failed. In the event the circuit outputs differ from the known good responses, the circuit is labeled as bad. Reference 8b gives a good overview of ATE. A block diagram of an automatic tester is shown in Fig. 7. Each of the major blocks will be discussed.

Circuit Under Test (CUT). The circuit under test (CUT) is the device that is under test. Testing can be conducted on the device at several points in the manufacturing process. For example, the device can be tested immediately after manufacture, before the wafer has been broken up into individual dies. At the other end of the spectrum, the circuit can be tested after the die has been packaged. The circuit can also be tested after it has been placed into service. Bed of nails testers have some capacity to test devices that are an integral part of the circuit board.

In order to achieve an accurate and reliable test, the tester must operate at the device speed and also check the timing of all parameters to specifications. In practice, the testers can test the device at speed; however, the testers can be extremely restricted in their capacity to check for exact timing. The problem lies in the manner in which outputs are checked. In most testers, the outputs are sampled at a regular rate. For example, the Teradyne model J971SP is capable of a peak data rate of 400 MHz, or 2.5 ns per pattern (http://www.teradyne.com/prods/std/j971/j971.html). In addition, a tester capable of taking samples at this rate must be capable of accepting the outputs and incorporating them in a process that can determine whether the device is defective or not.

Input Generator. The input generator can provide previously specified patterns or have some capacity to generate patterns. The simplest and most straightforward implementation is in the form of a memory that holds patterns that have been specified by a test engineer. The memory architecture must be designed to be able to supply patterns at a rate consistent with the test rate. Alternatively, the patterns applied to the CUT can adapt depending on the output responses of the CUT. For example, in the event the CUT is defective, an adaptive test can be applied to identify the specific defect. Lastly, patterns can be generated in a pseudorandom fashion. Pseudorandom pattern generation has the benefit of requiring simple circuitry that has provable probabilistic defect coverage for combinational circuits which is a function of the number of pseudorandom patterns that have been applied.

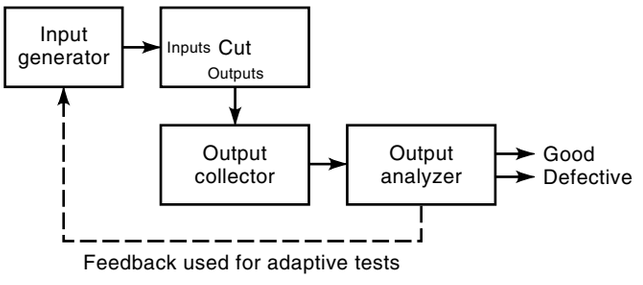

Figure 7. Block diagram of an automatic tester. The automatic tester is used to test integrated circuits.

Output Collector. The output collector collects the output results for subsequent passage to the output analyzer. One form of output collector is a simple pass through operation. In this form, the outputs of the circuit are passed to the output analyzer without modification. In some testing applications, the output collector compacts the output responses into a signature that is indicative of whether or not defects are present in the CUT. In many digital circuits, Linear Feedback Shift Registers (LFSRs) are used to compute the signatures. Other applications may utilized different structures to compute the signatures. For example, in Digital Signal Processing (DSP) applications, a digital integrator is shown to compute a signature with provable fault detection capabilities (9).

Output Analyzer. The purpose of the output analyzer is to determine whether the circuit is operating fundamentally defect free. In many automatic testers, the output analyzer consists of a memory containing all of the expected responses to the circuit. For each presentation of the input, the CUT outputs are compared to the previously determined good circuit outputs. A miscomparison results in the circuit being found to have one or more defects. The miscomparison usually does not indicate the specific location of the tests; however, additional tests may be required to identify the locations. In the event that the specific defect location is desired, additional likely adaptive tests would be necessary. Through a searching process, output results can be fed back to the input generator to emit tests consistent with the searching process. In the event output response compaction has been applied, relevant comparisons with the compacted signatures will be applied.

Parametric Testing. Parametric testing tests the electrical characteristics of the signals. For example, circuit outputs are generally required to be able to sink and source particular current levels while maintaining acceptable logic levels. Such testing requires measurement of both voltage and current parameters. In addition, testers have the capability to perform some timing tests that are in addition to performance related to device clock frequency.

Economics of Test

Not surprisingly, economics plays a role in how testing approaches are applied (3). In the manufacturing process, defects will occur, so it is impossible to ignore testing entirely. The costs include design, manufacturing, testing, and maintenance costs. On the design side, the engineer must decide what, if any, test structures to include in the design. If these

design structures are not in a library, they must be designed. In addition, the actual test regimen must be constructed. Complicating the matter, the costs associated with the test regimen can affect the costs associated with constructing the test set. Employing many design for test (DFT) approaches reduces the effort needed to develop test regimens. Since these costs are incurred in the design phase, they can be distributed over the manufacturing run, with a large manufacturing run minimizing the impact of the larger design costs.

The manufacturing costs are direct costs and are incurred with the manufacture of each device. In general, adding DFT capabilities increases the die size of the device, reducing the number of manufactured dies. Assuming standard yield models and assuming that no redundant capacity is included to compensate for increased die defect rates, the yield of the individual dies is reduced. Using highly simplistic assumptions, some useful observations can be made. First, the wafer is assumed to be 15 cm in diameter, the die area is 1 cm^2. This example assumes that the entire wafer area is available for dies makes no attempt to compensate for test dies, test regions, or dies on the perimeter. In this example, 706 dies can be manufactured on each wafer. Assuming DFT increases the die area by 10%, the number of dies on the wafer is reduced to 642. Assuming a fixed cost to manufacture each wafer, the apparent cost per die increases by 10%. More significantly, the defect yield models show that the fraction of good dies manufactured decreases nonlinearly with the increasing area. Assuming negative bilinear defect statistics and an average of 1 defect/cm^2, the yield of the 1 cm^2 die is 44.4% producing 313 working dies, while the yield of the 1.1 cm^2 is 41.6% producing 267 working dies. The cost per working die manufactured is 17% higher for the circuit that employs test circuitry.

Testing of the manufactured device is an essential aspect of the manufacture. The extensiveness of the testing performed and the design for test approaches used in the design can impact the testing cost. To determine whether or not the device is defect free, a series of test inputs are applied to the device and the responses are observed. In the event the observed behaviors differ, the device is marked as bad. DFT in some cases can simplify and reduce in number the set of test inputs and can reduce testing time proportionately. Furthermore, circuits with BIST capabilities can dramatically reduce or even eliminate the reliance on external test equipment. In addition, testing can be used to qualify a part according to a government or industry standard. For example, burn-in is the process of running a device just after manufacture in order to weed out the weaker parts. Failure of these parts is termed infant mortality due to the early failure and the increased failure rates of newly manufactured parts. Additional testing can be performed at the extremes of temperature, vibration, and humidity to receive qualification for devices used in military designs.

Maintenance costs are related to the costs of the device once it is placed in service. While the device is in service, device failures can cause the system within which it is contained to fail. Indeed, without some sort of built-in self-test (BIST) capability, determining the cause and compensating for the failure can be difficult. For a device within the system, determining the difference between defect and design flaw may be extremely difficult. Incorporation of a built-in test capability can greatly enhance the maintainability because the device, in effect, may be able to indicate its failure and simplify the task of the equipment maintainers.

DESIGN FOR TESTABILITY

Design for testability methods are used to simplify and enhance the testing process through analysis and application of methodologies to facilitate easier testing. The understanding of DFT requires background in several areas. These include understanding how circuits fail, the models for circuit failure, and general methods for identifying circuit failures. Automated testing can be performed in several fashions which can have a great impact on design (1,2). First, the test can be performed either on-line or off-line. In an off-line test, the system is taken out of service to perform testing. Once the system is out of service, the CUT is either placed in an ATE or BIST circuitry is enabled and the test is performed. BIST approaches will be discussed in detail in the next section. Once testing has been conducted, the system continues its normal function. An on-line test requires the system to remain in service. In one approach, the system is tested at idle times, where the system is operational but has no tasks to perform. This type of test is nonconcurrent, because the test and the circuit function are mutually exclusive. On-line concurrent testing requires testing to be performed concurrent with ordinary circuit function. A simple on-line testing approach is parity checks of transmitted data, however which may have unacceptable fault latencies. Special design techniques may be necessary to support on-line concurrent testing with high fault coverage that achieve low fault latencies.

Test Point Selection

A simple method for improving the testability of a design is through the addition of judiciously selected test points to serve as auxiliary points for controlling and observing internal nodes of the circuit. The identification of these points can follow from a testability analysis of the entire design by determining the difficulty with which internal points may be tested. Circuit notes that are among the most difficult to test are selected as the test points. As test points are identified, the testability analysis can be repeated to determine both how well the additional test points improve testability and to determine whether additional test points are necessary. The disadvantages to employing test points is that this approach is an ad hoc method for identifying test points and the test points will require additional inputs and outputs to the system.

Scan Design Methods

Scan design methods are DFT methods that build on the test point selection methodology by identifying specific circuit structures that would serve as suitable points for test points. Furthermore, these test points are interconnected as a large shift register enabling direct controllability and observability at all stages of the shift register. The scan design methods use either the flip-flops that are part of the design or other "geographic" clues such as system partitions to define the shift register paths. A key feature of scan design methods is that large sequential circuits are broken into smaller pieces that are more easily tested. Taken to its fullest extent, the

circuit is decomposed entirely into combinational and sequential parts. Historically, these approaches follow from techniques incorporated into the IBM System/360 where shift registers were employed to improve testability of the system (10). A typical application of a scan design is given in Fig. 8. Note the switching of the multiplexer at the flip-flop inputs controls whether the circuit is in test or normal operaiton. Differences between different scan design methods occur in the flip-flop characteristics or in clocking.

Scan Path Design. In scan path design (11), the circuit is designed to operate in one of two modes: normal and test. When the circuit operates in normal mode, all test circuitry is disabled and the circuit operates per its functional requirements. When in test mode, all flip-flops are reconfigured into a shift register whose contents can be scanned in or out through special test inputs and outputs to the circuit. The purpose of the shift register is to give direct controllability and observability to all the flip-flops in the shift register. In essence, the test mode decomposes the circuit into its sequential and combinational parts. The combinational logic can be tested with any of the well-known approaches for developing test patterns for combinational circuits. Relevant aspects of the design include the application of a race-free D flip-flop improving testability of the flip-flops.

Level Sensitive Scan Design. One long-standing and successful DFT approach is IBM's level sensitive scan design (LSSD) approach (12). Similar to the scan path approach, the circuit can be operated on one of two modes: normal and test. A key difference between the approaches is in the structure of the flip-flops. In LSSD, two clock phases are used to clock flip-flops to guarantee the detection of clock faults.

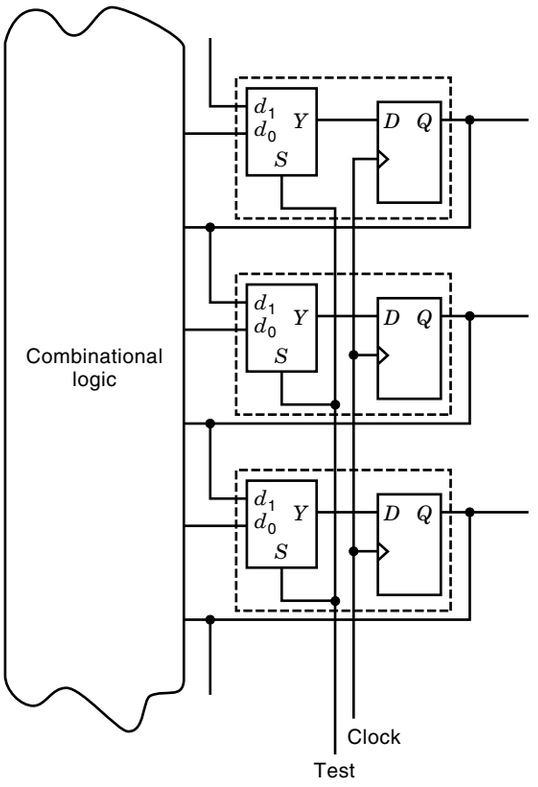

Figure 8. Scan path test structure. All flip-flops can be configured into an externally loadable shift register. In this configuration, combinational logic can be tested directly.

Boundary Scan Techniques. In many design approaches, the option of applying design for testability to some components is impossible. For example, standard parts that might be used in Printed Circuit Boards (PCBs) which are not typically designed with scan path in mind. As another example, more and more ASIC design consists of design using cores, subsystems designed by third party suppliers, in the design. The core subsystems are typically processors, memories, and other devices that until recently were individual VLSI circuits themselves. To enable testing in these situations, boundary scan methods were developed. Boundary scan techniques employ shift registers to achieve controllability and observability at the input/output parameters of circuit boards, chips, and cores. An important application of boundary scan approaches is to test the interconnect between chips and circuit boards that employ boundary scan techniques. In addition, the boundary scan techniques provide a minimal capability to perform defect testing of the components at the boundary. The interface to the boundary scan is a test access port (TAP) that enables setting and reading of the values at the boundary. In addition, the TAP may also allow internal testing of the components delimited by the boundary scan. Applications of boundary scan approaches include BIST applications (13), test of cores (14), and hierarchical circuits (15). The IEEE has created and approved the IEEE Std 1149.1 boundary scan standard (16). This standard encourages designers to employ boundary scan techniques by making possible testable designs constructed with subsystems from different companies that conform to the standard.

Built-In Self Test

Built-In Self Test (BIST) approaches add circuitry to a design to enable the circuit to test itself. The test can, depending on its design, be conducted autonomously or while the device is out of service. The necessity of requiring the device to test itself places constraints on the manner in which inputs are generated for the design and monitoring the outputs, when compared with a test used in conjunction with ATE. Furthermore, in cases where ATE testing is difficult, the device test can be composed of a combination of ATE testing and BIST approaches.

Running chip tests on all devices can be expensive, especially when yields are low. Effective application of BIST can enable circuits to assess their own health, reducing the amount of time necessary on expensive automatic test equipment. Furthermore, BIST enables systems in service to be configured to run automated tests. The benefit of this type of test is that systems can determine their health and report a failure possibly before problems appear at the system level. Second, for systems that have built-in redundancy, the results from an automated test can be used to switch-out a failed component or module and insert a new module.

BIST can require that the designer conform to a particular set of design rules and require additional circuitry be added to the system being designed. This can reduce design flexibility by restricting the types of designs that are possible as well

as potentially restricting the functionality of the design so that the entire system can fit in one device. Despite this, BIST approaches confer great benefits.

Test Pattern Generation and Built-In Test. The requirement for built-in test places great restrictions on the test generation process in two ways. First, the actual generation of test patterns must be self-contained within the circuitry itself. This implies the presence of circuitry that generates sequences of test patterns. While in theory, circuitry can be designed to produce any test pattern sequence, in practice the required circuit may be excessive or impossible to include. As a result, simpler circuitry must be employed in order to perform test pattern generation. Three classes of circuits are typically employed because they have good asymptotic performance and because they have the ability to produce any sequence of test patterns.

The first type of circuit is a simple counter. The counter can be constructed asynchronously and will thus only require as many flip-flops as there are test inputs. Counter solutions may be impractical for two reasons. First, if the number of test inputs are large, the time required to count through the entire sequence can be too long. Second the counter may be unable to test for other types of faults, such as delay faults or CMOS faults. Researchers have investigated reducing the count sequence so that more reasonable test lengths can be achieved (22). The second type of circuit generates pseudorandom sequences with LFSRs. Theoretically, as the number of random test patterns applied to the circuit increases, fault coverage increases asymptotically to 100%. Much research effort has gone into the development of efficient pseudorandom sequence generators. An excellent source on many aspects of pseudorandom techniques is (17). Third, a special circuit can be constructed that efficiently generates the test patterns. In this case, the desired sequence of test patterns are examined and a machine is synthesized to recreate the sequence. Memory tests have shown some success in using specialized test pattern generator circuits (27).

In order to determine whether a fault is present, the outputs of the circuit must be monitored and compared with outputs representative of fault-free behavior. Test pattern generation equipment solves this by storing the expected behaviors for the circuit given the sequence of inputs supplied by the tester. Similar to the problem of supplying test patterns in the context of automatic testing, it may be impractical to store or recreate the exact circuit responses. Duplication approaches (several are summarized in Ref. 4) can be employed to duplicate a subsystem to ensure that a copy of the expected circuits are available. By completely duplicating the system circuitry, the duplicated systems can be operated concurrently from the same set of inputs where any discrepancy between the duplicated systems results in the detection of a fault. While this approach may be applicable to systems requiring high reliability or fault tolerance, it would be an undesirable approach in most cases. Another widely used solution is to compress the circuit responses into a special code word, a signature, that is indicative of the presence or absence of faults. The signature represents the lossy compression of the circuit responses. The computation of the signature should achieve high probability of detecting the faults in the system while using the smallest quantity of resources to do so.

Methods for Compressing Test Data Sets. As indicated, test set compression is essential to producing economical test cases for use in automatic testing. Compression methods utilize test hardware in a fashion that retains much of the fault detection capabilities, while minimizing testing resources. The compaction process can be measured with respect to the following attributes (25): space, time, function specificity, linearity, and memory. The test data set can be represented as a two-dimensional matrix where each row is associated with one measurement point and each column corresponds to a test pattern that is applied at a particular time. Compression is a transformation that reduces the size of this matrix. The compression may or may not reduce the effectiveness of the overall test; however, a small reduction in test coverage is often acceptable. Space and time compression occur when the number of rows and/or columns are reduced. The compression can be expressed mathematically as $D = \Phi C$, where D is the original test data matrix, C is the compressed test data matrix, and Φ is the compression transformation. The manner in which the transformation Φ is selected affects the matrix size as illustrated, but it may have implications in regards to the remaining three attributes. Function specificity occurs when the transformation relates the number of compactor stages to either the number of inputs or test patterns, or also if the compaction is related to the sequences of binary patterns or inputs. Linearity is a property that follows from using the exclusive OR function to derive Φ, which can be shown to be a linear operator in the finite field GF(2) (26). For example, systems such as the LFSR would be considered a linear time compactor. Memory is the property that a bit in D is a function of both past and present bits in C.

For example, space compression measures the reduction in test data width. For example, consider a test strategy constructed that uses 100 test points. Suppose further that the values from pairs of test points are exclusive or gates are exclusive together. If the resulting 50 signals are monitored, a space compaction of 50% is realized. Time compaction occurs when the number of columns is reduced by some process. For example, SISRs and MISRs can be used to compress the number of columns to 1.

Signature Analysis. Signature analysis is the process of compressing output responses frequently used in BIST with a Single Input Shift Register (SISR) or Multiple Input Shift Register (MISR) to compute a signature, which is a compression of the output responses (17). The signature is a quantity that is representative of the condition of the system. Figure 9 gives the architecture of SISRs and MISRs. A signature for

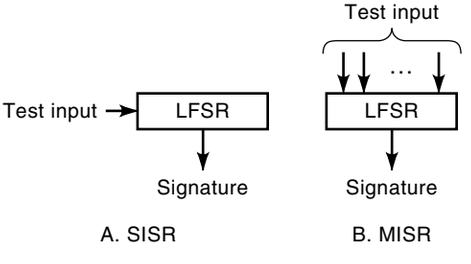

Figure 9. Signature generation architectures. The signature can be computed by sampling one signal or several at the same time.

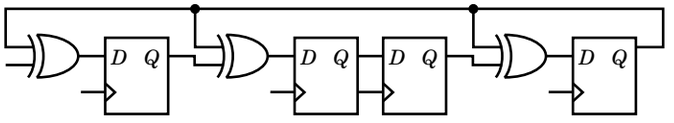

Figure 10. Linear feedback shift register. The LFSR is a key building block in the implementation of signature generation, pseudorandom pattern generation, and code generators/checkers.

the known good circuit is compared with the signature for the CUT where any discrepancies indicate the presence of a defect. SISR and MISR are linear sequential compaction schemes of the observed test point(s). The architecture of SISR and MISR are LFSRs at the core which are both forms of LSFRs. The signature, being a compression of output responses, gives no indication of the failure. Furthermore, the number of bits in the signature is far less than the number of test patterns. As a result, several different circuit conditions can result in the same signature, termed *aliasing*. In practice, if a fault is present, it is desired that the signature produced differs from the fault-free case. It is possible in some designs for the signature to be the same even though it was the result of compaction of an entirely different set of output results. In Ref. 18, the aliasing probability upper bounds were derived for signatures computed with SISRs. In addition in Ref. 19, methods for developing MISRs with no aliasing for single faults were developed.

LFSRs have properties that are useful for generating effective signatures. First, the LFSR approach offers a simple and effective compaction approach for the detection of faults in the circuit. The LFSR consists of some number of flip-flops, N, an linear connections fed back to stages nearer the beginning of the shift register. The general structure of the LFSR used as an SISR is shown in Fig. 10. In addition, other types of circuits have shown the ability to effectively compute signatures. For example, DSP often results in modular design constructed from accepted DSP building blocks. In particular, digital integrator circuits have been employed to compute signatures (9).

BILBO. The built-in logic block observer (BILBO) approach has gained a fairly wide usage as a result of its modularity (28). The BILBO approach is derived from scan path approach where the scan circuitry and other functions are encapsulated in a BILBO register. In addition, connections within the BILBO register enable computation of a signature, suitable for fault detection. A four-bit BILBO register is shown in Fig. 11. BILBO registers operate in one of four modes. The first mode is used to hold the state for the circuitry as D flip-flops. In the second mode, the BILBO register can be configured as a shift register that can be used to scan values into the register. In the third mode, the register operates as a multiple input signature register (MISR). In the fourth mode, the register operates as a parallel random pattern generator (PRPG). These four modes make possible several test capabilities.

One example application of BILBO registers is shown in Fig. 12. In test mode, two BILBO registers are configured to isolate one combinational logic block. The BILBO register at the input, R_1, is configured as a PRPG, while the register at the output, R_2, is configured as a MISR. In operation, for each random pattern generated, one output is taken and used to compute the next intermediate signature in the MISR. When all tests have been conducted, the signature is read and compared with the known good signature. Any deviation indicates the presence of faults in the combinational circuit. In order to test the other combinational logic block, the functions of R_1 and R_2 need only be swapped. Configuration of the data path to support test using BILBO registers is best achieved by performing register allocation and data path design with testability in mind (29).

Memory BIST. Semiconductor memories are regular structures manufactured with the goal of maximizing the capacity for a given area of silicon. While in principle the memories can be tested as other sequential storage elements, in reality the overhead associated with utilizing scan path and similar test approaches would severely impact the storage capacity of the devices. The regularity of the structure, however, gives the capability of testing the memory with hardware structures outside the memory arrays proper. Among the test design considerations of memories is the number of tests as a function of the memory capacity. For example, a test methodology was developed (27) for creating test patterns. These test patterns could be computed using a state machine that is relatively simple and straightforward. The resulting state machine was shown to be implementable in random logic and as a microcode driven sequencer.

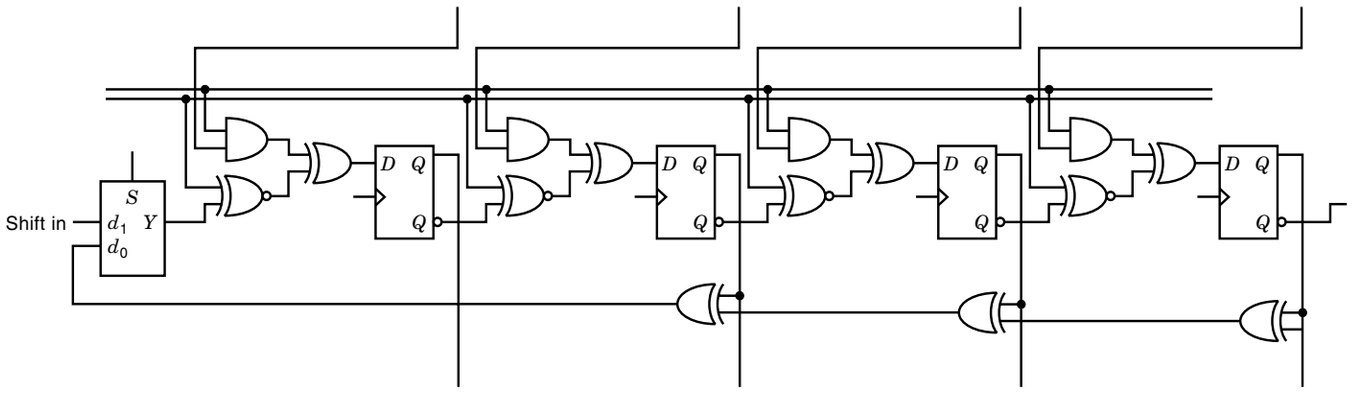

Figure 11. Four-bit BILBO register. The BILBO register can operate in one of four modes: simple register, shift register, pseudorandom pattern generator, or signature register.

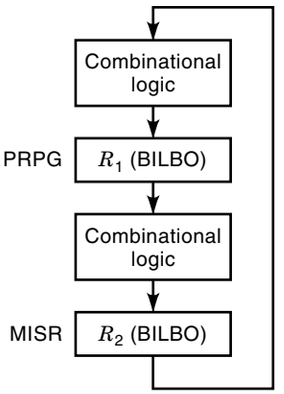

Figure 12. Example BILBO architecture. BILBO registers are used in pairs to isolate combinational logic functions.

Programmable Logic Arrays

With the increasing complexities of programmable logic arrays (PLAs) with sizes approaching 1 million devices, PLA test is becoming an important part of the design process. PLA test can be viewed from two perspectives. First, the "blank" device can be tested and deemed suitable for use in an application. Second, the programmed device may be subject to the testing process, itself.

Self-Checking Circuits

Self-checking circuits (SCCs) are circuits that, by their construction, have the inherent capability to detect whether a failure exists in their circuitry (4). Self-checking circuits are not employed in typical applications because of the overhead required which can be slightly more than 50% additional circuitry. SCCs enable automatic testing while the circuit is operational and can achieve high fault coverage without extensive test structures or regimens. The key feature of SCCs is that they employ complementary circuits and convey information using a codeword consisting of two bits. The pair of bits represents the true and complementary values of the signal being transmitted. Any fault results in the pair of signals being identical, allowing clear detection of the fault. A simple example of such a SCC is shown in Fig. 13. The dual of a function is defined in terms of the original function in the following way

$$f_{\text{dual}}(x_1, x_1, \ldots, x_n) = \overline{f}(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n) \tag{1}$$

In short, the dual outputs the complement of the desired function given the complements of the inputs. In ASIC development, SCCs provide two levels of protection and detection of faults. First, if either the function or its dual has fault, then only that function is affected. As a result, if the fault is present, either the circuit will provide the correct output or it will give the incorrect output. In the event the incorrect output is supplied, an error will be detected because the dual function will still be providing the correct output. Second, in the event of a failure in the design tools, the function and its dual may be affected in different ways so that a tool failure will have a higher probability of detection. The final issue that must be addressed in SCCs is the checking circuitry. The checking circuitry must be designed to both detect incidences of illegal signal combinations and be self-checking, itself. SCCs are use in applications where fault detection is essential to the completion of the mission and are rarely used in common applications. Cases where SCCs might be used are space and aerospace applications.

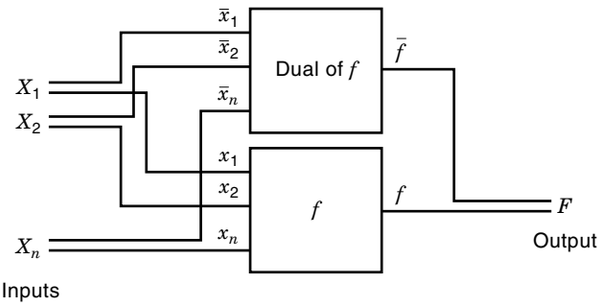

Figure 13. A self-checking circuit. Self-checking circuits are composed of complementary circuits of which only one malfunctions when a defect is present.

TEST PATTERN GENERATION

Two general approaches can be used for developing tests that achieve high fault coverage. The first approach employs functional test and the second specifically targets defect tests. A test is an input or sequence of inputs that can be used to detect a fault in a circuit. For example, in Fig. 14, a test for *G SA*-1 is *ABCDEF* = 110*X*0*X*, where *X* is a don't care. Note that by setting *AB* = 11, the expected value on node *G* is 0, while with the fault present the fault value is 1. In addition, setting *CD* = 0*X* and *EF* = 0*X* enables the propagation of the discrepancy due to the fault to the output of G_4, where it can be detected by examining node *J*.

Test pattern generation is the process of determining a set of test patterns that result in an effective test giving an acceptable test coverage (1). A good test requires testing of all aspects of the circuit for as many defects as are reasonably possible. The creation of the test is very much dependent on the type of defect model that is assumed for the device. For example, the stuck fault model may be used to developed test patterns. Even if the test engineer created a test that covered 100% of all stuck faults, defective CMOS devices may pass the test because the actual defects in the technology tend to show delay or bridging faults.

Test patterns can be created in several ways. First, perhaps most intuitive source of test patterns come from func-

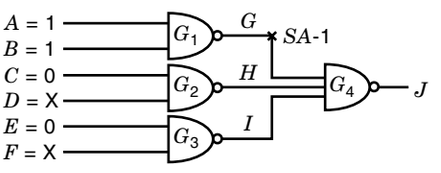

Figure 14. Detection of fault in combinational circuit. The input ABCDEF = 110X0X detects the G stuck-at 1 fault.

tional tests of the system under consideration. The second source of test patterns comes from a concerted attempt to exhaustively test for every defect in the circuit. The third approach is to create a random set of test patterns. Designs are often supported with DFT methodologies such as scan registers to decompose the system into sequential and combinational parts that can be tested largely independent of one another. The sequential part is often configured as a shift register which is straightforward to test. As a result, much effort has been expended on developing test patterns for combinational circuits. Several test generation approaches are immediately discussed.

Functional Test

A functional test can be used to determine whether the circuit meets computational and functional requirements. In addition, by exploiting known circuit structures with known testability characteristics. A designer will construct these tests in the process of debugging the design so that the functional characteristics of the system can be demonstrated. In theory, functional tests can be constructed to achieve very nearly 100% fault coverage; however, the number of tests may be prohibitive. For example, a functional test of a multiplication circuit might include tests computing several products. Functional tests can be used alone or in conjunction with defect tests to achieve the desired fault coverage. In the event DFT circuitry is incorporated, functional tests verify that the circuit function is achieved by operation in its normal mode by deactivating test circuitry.

Defect Tests

In defect testing, it is taken for granted that the system design meets functional requirements. In general, the goal of defect testing is to test and verify that all circuit structures and components are operating defect free and within manufactured tolerances. As a result, the actual test patterns presented to the system may bear no resemblance to circuit inputs that would ordinarily be encountered in the field.

Defect tests can be constructed in an ad hoc fashion, provided the appropriate tools are available, such as an accurate fault simulator. Constructing effective tests can be eased with the introduction of DFT methods that improve the controllability and observability of internal circuit structures. For example, the introduction of test points and scan registers can aid the designer in efficiently achieving acceptable defect tests.

In addition, algorithms exist for automatically determining a set of defect tests that under certain conditions and assumptions, can achieve 100% fault coverage. One such algorithm is the *D*-algorithm (20,21). The *D*-algorithm can be used to determine test vectors for combinational logic and achieves 100% fault coverage for such circuits, provided the circuits do not contain any redundancy, as what might be present for eliminating hazards or to achieve fault tolerance. The *D*-algorithm can be effectively applied to circuits that include a test configuration where all flip-flops are clocked by the same clock signal, test circuitry is present that reconfigures the circuit so that all flip-flops are a part of a long shift register, and a mechanism for shifting values into and out of the shift register are present. The test patterns can be used for BIST applications provided the patterns can be efficiently stored or recreated by test circuitry.

Lastly, pseudorandom tests can be constructed by randomly creating test patterns that are applied to the circuit. The fault coverage is a function of the circuit and the number of different test patterns that are applied. Furthermore, the coverage asymptotically approaches 100% as the number of different test patterns that have been applied increases. The benefit of pseudorandom tests is that the tests can be easily created with simple circuit, desirable for BIST applications.

Exhaustive Testing

Exhaustive testing is the testing under all possible operating conditions for all faults. Because each possible operating condition is targeted all faults are guaranteed to be detected, provided they can be detected by any test. The problem with exhaustive testing is that the number of tests required to achieve exhaustive testing is in most cases intractably large. For example, an irredundant combinational circuit with four inputs may be exhaustively tested for all stuck faults with 16 test patterns, one for each input combination. A circuit of 64 inputs would require $2^{64} = 1.8 \times 10^{19}$ tests to be similarly tested. Given a tester that could present 10^9 patterns per second, the test would require over 500 years. Further complicating the process of exhaustive testing is the presence of sequential circuits and also several failure modes that require two or more consecutive patterns to test for individual faults.

Pseudoexhaustive Testing

In pseudoexhaustive testing, the circuit is partitioned into component blocks that can be tested exhaustively. This approach achieves some benefits of exhaustive testing in that all faults of the desired class are tested while reducing the scope of the test to a subcircuit of a more manageable size. In addition, counters (22) and LFSR circuits (17) are capable of generating exhaustive input patterns for some fault models. The circuit can be partitioned based on several criteria. One criteria for partitioning is to examine the dependence of outputs on the inputs. If an output is dependent on fewer than the total number of inputs, then select these inputs as input to the pseudorandom test (2).

The *D*-Algorithm

The *D*-Algorithm (20,21) is a test pattern generation algorithm that can be used to determine the set of test patterns to detect all detectable stuck faults in combinational circuits. While the types of circuits that are used in digital systems are not combinational, these circuits can be decomposed into combinational parts using scan design methodologies. The "*D*" in the *D*-Algorithm represents a discrepancy between the fault free and faulty circuit, where *D* is used both to represent the fault for which a test pattern is desired as well as the propagation of the discrepancy to the output of the circuit for detection of the fault. In the circuits, D and $\overline{D}$ will be used as logic values with the familiar 1 and 0 logic values. In the context of fault detection, D represents a discrepancy where the circuit node should be 1 yet is 0 due to the presence of a fault. In a complementary fashion, $\overline{D}$ represents a discrepancy where the circuit node should be 0 yet is 1 due to the presence of a fault. The theme of the algorithm is to assign D and $\overline{D}$ to

circuit nodes to represent faults. Next, the discrepancy is propagated to the outputs by sensitizing gates to enable propagation of the discrepancy along all paths until the discrepancy appears at the circuit output(s). This process is termed the *D*-drive. Next, inputs of the remaining gates are set to support the propagation of the discrepancy to the circuit outputs which is termed the consistency operation. Backtracking in the consistency operation occurs whenever a node is required to be both 1 and 0. Among the significant features of the *D*-Algorithm is the ability to find a stuck fault test if one exists. In addition, the *D*-Algorithm can be extended to test for delay faults and nonfeedback bridging faults (21).

Pseudorandom Test Pattern Generation

While the *D*-Algorithm can be used to determine the test patterns for a particular circuit, as a result of searching, the *D*-Algorithm can be computationally expensive. A computationally less expensive yet but undirected approach uses test patterns generated in a pseudorandom fashion (17). The basic idea behind Parallel Random Pattern Generator (PRPG) is the assumption that many approximately equally desirable tests exist for a circuit. By randomly sampling the individual tests, probabilistically speaking, an acceptable test set can be found. LFSRs are simple circuits that can be used to generate pseudorandom test patterns.

Weighted Test Patterns

Test patterns constructed using PRPGs result in patterns where each bit has an equal chance of being either 1 or 0. For some circuits, this uniformity in the test patterns may not result in an effective set of tests. In particular, certain types of patterns are more productive in terms of the quantity of faults that can be detected. Adders and counters, for example, are effectively tested when patterns result in either carries generated or when carries could be generated if faults are present. One of the first applications of weighted test patterns assumed that the circuit had no DFT circuitry and that the tester changed one input at time. After presenting random set of test patterns, it was observed that some input changes resulted in a larger number of devices in the circuit changing. By changing the statistics of the inputs and making the inputs change in a pseudorandom frequency dictated by their importance, measured by internal gate signal changes, the benefits of weighted test patterns were demonstrated (23).

Multiple Pattern Methods

Under certain circumstances, a single pattern applied to a circuit is insufficient to test for some types of faults or to achieve certain testing goals. Multiple test patterns are necessary when it is desired to locate the pattern. Since one test pattern will detect the presence of several faults, additional tests are required to rule out other defects. In addition, delay faults require two test patterns to detect delay faults. The first pattern sets up the conditions and the second pattern makes the presence of the fault visible. The *D*-Algorithm can be easily enhanced to test for delay faults (21). In CMOS circuits, transistor faults can result in a combinational circuit operating as a sequential circuit. To detect this fault, the first pattern stores a value and the second pattern propagates the discrepancy to the output of the circuit (24).

Test Pattern Generation for PLAs

As indicated earlier, PLAs are susceptible to crosspoint faults where the function programmed into the PLA is changed by the addition or omission of a term in the AND array or OR array. These faults do not fit the standard stuck fault model, so a different strategy to test for crosspoint faults must be adopted.

CAD TOOLS

Support of automatic testing in the CAD tool is as important as understanding the test techniques given the high density and complexity of today's digital systems. The inclusion of test support in the CAD tool facilitates integrating good test practices into the design process. Tool builders have recognized the importance of including testing approaches. Two leading companies in this respect are LogicVision, Inc. and Mentor Graphics. Furthermore, the development of hardware description languages that has resulted the widely used Verilog and VHDL greatly simplifies the process of adding automatic testing capabilities. VHDL is the VHSIC Hardware Description Language where VHSIC is very high speed integrated circuits (32).

Reference 30 gives a good summary of CAD approaches.

Fault Simulation

Fault simulation is a simulation capable of determining whether a set of tests can detect the presence of faults within the circuit. A fault simulator is a logic simulator with the capability of keeping track of whether one or more faults can be detected given the inputs to the circuit. In practice, a fault simulator simultaneously simulates the fault-free circuit concurrently with circuits having faults. In the event that faults produce circuit responses that differ from the fault-free cases, the fault simulator records the fault that was detected.

In order to validate a testing approach, fault simulation is employed to determine the efficacy of the test. Fault simulation can be used to validate the success of a test regimen and give a measure fault coverage achieved in the test. In addition, test engineers can use fault simulation for developing functional test patterns. By examining the faults covered, the test engineer can identify circuit structures that have not received adequate coverage and target these structures for more intensive tests.

In order to assess different fault models, the fault simulator must be able to both model the effect of the faults and also to report the faults that were detected by a particular input sequence. In the test for bridging faults detectable by I_{DDQ} testing, traditional logic and fault simulators are incapable of detecting such faults because these faults may not produce a fault value that can differentiate faulty from fault-free instances. In Ref. 31, a fault simulator capable of detecting I_{DDQ} faults is described.

Test Structure Libraries

By incorporating DFT structures and techniques in libraries, the designer is given the ability to easily include accepted testing practices. Some testing practices can be incorporated in a fashion that is to some extent transparent to the design engineer. In doing so, DFT techniques can become a more in-

tegral part of the design process. For example, scan design approaches can be incorporated in a design provided all flip-flops operate off the same clock. Connecting flip-flops in a scan path architecture can be an option with specific details of the test structure being hidden from the designer unless it is immediately relevant to the design. Languages such as VHDL can offer DFT capabilities with the addition of libraries (32).

MIXED-SIGNAL TESTING

Analog systems have increased in complexity along with digital systems. With increased system complexity comes with it the necessity to perform defect and parameter testing. Furthermore, since both analog and digital circuitry require essentially the same manufacturing process, a common practice is to include both analog and digital circuitry in the same chip. With the increasing complexity of analog circuits and with the introduction of digital circuitry, testing becomes a more important issue. Analog circuitry presents special problems in testing because analog signals are continuous, making it impossible to develop a test for all possible cases. Second, complex digital circuitry can be included in a circuit that contains analog circuitry for which testing is desired. At the interface, the digital signals serve as actuators to the analog functionality. In general, however, ABIST (Analog Built-In Self Test) shares many of the same features and challenges of digital BIST. First, for nonconcurrent testing, the CUT must be switched to a special test mode where inputs are generated and output responses are compared with accepted norms for behavior. At the system level, ABIST is not unlike digital BIST. The CUT requires an input stimulus and the outputs must be monitored and compared against a measure indicative of the CUT fault status.

Signal Generation

Among the challenges of developing ABIST architectures is having the capability of injecting a test signal into the CUT (33). Digital systems are straightforward from the point of view that in principal, any pattern can be generated for input. Analog signals can take on an infinity of values and combinations. Indeed, it is impossible to test an analog device for all possible operating conditions and levels. With the selection of appropriate signals, a fairly complete test is possible. Ideally, having the signal generation capabilities of a signal generator from the lab bench along with the capability of programming the signal generator gives the ideal combination of capabilities for testing analog devices. Integrating these capabilities on analog chips, however, can be impractical due to the difficulty of creating a signal generator with sufficient capabilities to test the CUT. The simplest form of signal generator simply generates a sine wave with either fixed or variable frequencies. Such a signal generator is termed a *single-tone signal generator*. The frequency response of an analog device can be obtained with a programmable single-tone signal generator that scans the desired range of signals in order to obtain a frequency response. This approach can be ineffective at generating tests of nonlinear properties. For example, a two-tone signal generator can be used to test for nonlinearities by determining the magnitude of frequencies that result from mixing the original two tones.

Mixed Testing

In some analog applications, both analog and digital circuits are present. For example, an A/D converter requires analog circuits to sample the analog input compare the analog signal levels to internal references. Digital circuits are then used to take the results of these comparisons and output a binary integer representative of the analog voltage level. For a complete test, both the analog and digital circuits must be tested, as well as the interfaces between the two. Parts of digital circuits can be tested using the techniques of purely digital circuits. In Ref. 34, a BIST approach was developed that incorporated digital DSP hardware to perform frequency response and other parameters of an analog to digital converter.

AUTOMATIC TEST EXAMPLES

Two examples of microprocessors that employ automatic testing methodologies are presented in this section. It is interesting to note the range of techniques that are employed even in the same design. Microprocessor CPUs present special challenges for automatic testing as a result of their enormous complexity, state of the art implementation methodologies, and the complications of mass production. Indeed, because microprocessors are manufactured in volume, the economics of employing testing methodologies are carefully weighed against the benefits. In today's modern microprocessors, subsystem structures of every type, each with its own testing challenges, coexist and must be tested. The techniques employ a combination of both built-in test and the application of ATE.

Failure Analysis in Intel Pentium and Pentium Pro Processors

Intel incorporated several DFT approaches that enable device test and manufacturing failure analysis (35). Failure analysis is necessary to determine the cause of device failure. In a manufacturing setting, specific patterns of failures may be indicative of problems with the process. Thus, the identification of the specific failure mode is essential for adjusting the manufacturing process to reduce the incidence of a specific type of failure. Approaches such as *e*-beam probing can identify failures, but Intel engineers recognized that current manufacturing technologies reduced the effectiveness of *e*-beam probing for several reaons. First, the component geometries were becoming too small to be resolved. Second, layers of metalization impedes observation of signals at the first and second layers of metalizations. Third, flip-chip packaging complicates *e*-beam probing.

Several DFT techniques were employed in the Intel Pentium processor in order to facilitate testing off the manufacturing line. The Intel Pentium has microcode that can be patched with externally loaded microinstructions, accomplished by down-loading the new microinstructions into a special memory. Special control logic transfers control to the microinstruction patches at desired points in the microprogram. The microinstructions are coded to aid in localizing the source of defects and can be used in conjunction with external probing methodologies. Another Pentium DFT capability allows external dumping of the contents of memory arrays, enhancing the observability of memories and control logic. Complementing these DFT techniques is a scan-out mechanism en-

abling the sampling of internal control and datapath signals. The cache memories also support a direct access test mode whereby the memories are accessed as a pipelined memory. A more detailed test mode, the low yield analysis mode, enables extraction of dc parameters for individual memory cells that are useful in failure analysis.

Dec Alpha Processor

Several testing methodologies were employed in the Dec Alpha Processor to achieve several goals (36). First, the Alpha was designed to achieve extremely high performance, and thus the testing methodologies had to be incorporated in such a way to not significantly affect performance. Second, testability methodoligies were employed to ease the burden on chip tester technology. Third, the testability methodologies were designed to work in conjunction with repair methodologies of chip memories. On the Alpha, it was decided not use full scan path methodologies as a result of the size of the processor and because some of the logic was implemented using dynamic logic. It was further recognized that the instruction cache could be used to store programs valuable in the testing process. Upon initial wafer probe, the Alpha instruction cache was designed to undergo a built-in self-repair operation prior to being loaded with a special test program. The purpose of the test program is to determine failed bits in the cache memories and report failures for their subsequent masking through laser repair. In addition, the Alpha has a manufacturing test port that supports limited internal scan path capabilities the form of 27 linear feedback shift registers. A mostly IEEE 1149.1 compatible TAP enables boundary scan test capabilities.

EMERGING TECHNOLOGIES

I_{DDQ} Test Patterns

In I_{DDQ} testing, the current profiles for CMOS circuits are used to test for defects in the circuits (7). CMOS circuits are particularly susceptible to bridging faults that are not always testable when developing tests according to the stuck fault model. I_{DDQ} testing is generally considered to be an augmentation to other testing approaches and has been shown to improve fault coverage. In I_{DDQ} testing, as test patterns are presented to the circuit, the quiescent current is monitored. If the current exceeds some threshold, defects are presumed to be present. Test patterns for I_{DDQ} testing can be the same patterns used to perform fault testing according to other models, augmented with current profiles for the different patterns in the test set. Because CMOS circuits can draw different quiescent currents depending on the internal function of the circuit, adopting a threshold for each test will give a more accurate test. In addition, test patterns can be created specifically for I_{DDQ} testing.

Challenges of Core-Based Design

In core-based designs, VLSI systems are constructed using complex components from libraries (37). Using simple to moderate complexity parts from a library has been a part of the industry for some time. The difference is that today, library components are of the complexity of entire systems from the past, such as CPUs and memories. Since the cores are supplied by third parties who consider their cores to be intellectual property, supplying enough information to effectively test the cores would require the disclosure of the design. It is impossible to be able to construct an effective test regimen without some knowledge of the internal structure of the core. Along with the core, the core vendors will supply test vectors that test the core to achieve a given level of defect coverage. The difficulty is that this requires the test patterns be applied directly to the inputs and outputs of the core, which can be embedded deep within the design (14).

Synthesis for DFT

In many ways, effective application of testing methodologies requires attention at every stage of the design process (29). While the focus of testing is on the individual components and gates in the circuit, initially, the designer will rarely work in this domain, given the complexity of most designs produced today. More typically the designer will design at the behavioral level. Synthesis for DFT can facilitate the design of a testable circuit in several ways. First, in synthesis, test structures can transparently be compiled into the design making it more testable. Second, by adding datapaths that improve controllability and observability, the testability of the design can be improved. In Ref. 29, resource scheduling is used to enhance testability. Four rules were introduced to improve testability which can be applied to the design at the behavioral level.

Rule 1: *Whenever possible, allocate a register to at least one primary input or primary output.*

Rule 1 improves the controllability and observability of the design.

Rule 2: *Reduce the sequential depth from an input register to an output register.*

In Rule 2, the paths through which data is processed are designed so that the data is stored in no intermediate registers. In doing so, improved controllability and observability of a design results.

Rule 3: *Reduce sequential loops by*
—*proper resource sharing to avoid creating sequential loops for acyclic data flow graphs, and*
—*assign IO registers to break sequential loops in cyclic data flow graphs.*

Sequential data path loops reduce the testability of the circuit. By avoiding the creation of sequential loops and breaking sequential loops that are unavoidable, testability is improved.

Rule 4: *Schedule operations to support the application of Rules 1 to 3.*

In Ref. 29, application of Rules 1 to 4 are shown to significantly improve the testability of circuits compared to cases that do not employ these rules. Interestingly, this approach does not require test circuitry traditionally used to improve testability, such as the application of scan design approaches.

SUMMARY

In this article, many aspects of automatic test were investigated. Automatic testing can be conducted by either external test equipment or can be incorporated in the circuitry of the system. Important issues in developing and using automatic test include knowledge of the types of failures that can be expected in the application technology. The types of failures can have a great impact on both the methodology for generating tests and also on the design of the systems. With the complexity of today's VLSI circuits, many circuits are incorporating DFT and BIST approaches to ease testing burdens.

APPENDIX 1. ACRONYMS AND ABBREVIATIONS

ABIST	Analog Built-in Self-test
ATE	Automatic Test Equipment
BILBO	Built-in Logic Block Observer
BIST	Built-in Self-test
CUT	Circuit Under Test
DFT	Design for Test
DSP	Digital Signal Processing
LFSR	Linear Feedback Shift Register
LSSD	Level Sensitive Scan Design
MISR	Multiple Input Shift Register
PCB	Printed Circuit Board
PLA	Programmable Logic Array
PRPG	Parallel Random Pattern Generator
SCC	Self-Checking Circuits
SISR	Single Input Shift Register
TAP	Test Access Port

BIBLIOGRAPHY

1. M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design,* IEEE rev. ed., Piscataway, NJ: IEEE Press, 1990.
2. P. K. Lala, *Digital Circuit Testing and Testability,* San Diego, CA: Academic Press, 1997.
3. J. M. Miranda, A BIST and boundary-scan economics framework, *IEEE Des. Test Comput.,* **14** (3): 17–23, July–Sept 1997.
4. B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems,* Reading, MA: Addison-Wesley, 1989.
5. M. Sivaraman and A. J. Strojwas, *A Unified Approach for Timing Verification and Delay Fault Testing,* Boston: Kluwer, 1998.

5a. N. K. Jha and S. Kindu, *Testing and Reliable Design of CMOS Circuits,* Boston: Kluwer, 1992.

6. J. Gailay, Y. Crouzet, and M. Vergniault, Physical versus logical fault models in MOS LSI circuits: Impact on their testability, *IEEE Trans. Comput.,* **29**: 1286–1293, 1980.
7. C. F. Hawkins et al., Quiescent power supply current measurement for CMOS IC defect detection, *IEEE Trans. Ind. Electron.,* **36**: 211–218, 1989.
8. R. Dekker, F. Beenker, and L. Thijssen, A realistic fault model and test algorithms for static random access memories, *IEEE Trans. Comput.-Aided Des.,* **9**: 567–572, 1996.

8a. K. M. Butler and M. R. Mercer, *Assessing Fault Model and Test Quality,* Boston: Kluwer, 1992.

8b. K. Brindley, *Automatic Test Equipment,* Oxford: Newnes, 1991.

9. J. Rajski and J. Tyszer, The analysis of digital integrators for test response compaction, *IEEE Trans. Circuits Syst. II, Analog Digit.,* **39**: 293–301, 1992.
10. W. C. Carter et al., Design of serviceability features for the IBM system/360, *IBM J. Res. Develop.,* 115–126, July 1964.
11. S. Funatsu, N. Wakatsuki, and T. Arima, Test generation systems in Japan, *Proc. 12th Des. Automation Conf.,* 1975, pp. 114–122.
12. E. B. Eichelberger and T. W. Williams, A logic design structure for LSI testability, *Proc. 14th Des. Automation Conf.,* New Orleans, LA, 1977, pp. 462–468.
13. A. S. M. Hassan et al., BIST of PCB interconnects using boundary-scan architecture, *IEEE Trans. Comput.,* **41**: 1278–1288, Oct 1992.
14. N. A. Touba and B. Pouya, Using partial isolation rings to test core-based designs, *IEEE Des. Test Comput.,* **14** (4): 52–59, Oct–Dec 1997.
15. Y. Zorian, A structured testability approach for multi-chip modules based on BIST and boundary-scan, *IEEE Trans. Compon. Packag. Manuf. Technol. B, Adv. Packag.,* **17**: 283–290, 1994.
16. IEEE, *IEEE Standard Test Access Port and Boundary-Scan Architecture,* Piscataway, NJ: IEEE Press, 1990.
17. P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques,* New York: Wiley, 1987.
18. S. Feng et al., On the maximum value of aliasing probabilities for single input signature registers, *IEEE Trans. Comput.,* **44**: 1265–1274, 1995.
19. M. Lempel and S. K. Gupta, Zero aliasing for modeled faults, *IEEE Trans. Comput.,* **44**: 1265–1274, 1995.
20. J. Paul Roth, Diagnosis of automata failures: A calculus and a method, *IBM J. Res. Develop.,* 277–291, 1966.
21. J. Paul Roth, *Computer Logic, Testing, and Verification,* Potomac, MD: Computer Science, 1980.
22. D. Kagaris, S. Tragoudas, and A. Majumdar, On the use of counters for reproducing deterministic test sets, *IEEE Trans. Comput.,* **45**: 1405–1419, 1996.
23. H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, The weighted random test-pattern generator, *IEEE Trans. Comput.,* **C-24**: 695–700, 1973.
24. BIST Test Pattern Generators for Two-Pattern Testing Theory and Design Algorithms, C. Chan and S. K. Gupta, *IEEE Trans. Comput.,* **45**: 257–269, 1996.
25. A. Ivanov, B. K. Tsuji, and Y. Zorian, Programmable BIST space compactors, *IEEE Trans. Comput.,* **45**: 1393–1404, 1996.
26. D. Green, *Modern Logic Design,* Reading, MA: Addison-Wesley, 1986.
27. M. Franklin, K. K. Saluja, and K. Kinoshita, A built-in self-test algorithm for row/column pattern sensitive faults in RAMs, *IEEE J. Solid-State Circuits,* **25**: 514–524, 1990.
28. B. Koenemann, B. J. Mucha, and G. Zwiehoff, Built-in test for complex digital integrated circuits, *IEEE J. Solid State Circuits,* **SC-15**: 315–318, 1980.
29. M. Tien-Chien Lee, *High-Level Tests Synthesis of Digital VLSI Circuits,* Boston: Artech House, 1997.
30. M. Gössel and S. Graf, *Error Detection Circuits,* New York: McGraw-Hill, 1993.
31. S. Chakravarty and P. J. Thadikaran, Simulation and generation of IDDQ tests for bridging faults in combinational circuits, *IEEE Trans. Comput.,* **45**: 1131–1140, 1996.
32. C. H. Roth, Jr., *Digital Systems Design Using VHDL,* Boston: PWS Publishing Company, 1998.
33. G. W. Roberts and A. K. Lu, *Analog Signal Generation for Built-In-Self-Test of Mixed-Signal Integated Circuits,* Norwell, MA: Kluwer, 1995.
34. M. F. Toner and G. W. Roberts, A frequency response, harmonic distortion, and intermodulation distortion test for BIST of a

sigma-delta ADC, *IEEE Trans. Circuits Syst. II, Analog Digit.,* **43**: 608–613, 1992.

35. Y. E. Hong et al., An overview of advanced failure analysis techniques for Pentium and Pentium Pro microprocessors, *Intel Technology J.,* **2**, 1998.
36. D. K. Bhavsar and J. H. Edmondson, Alpha 21164 testability strategy, *IEEE Des. Test Comput.,* **14** (1): 25–33, 1997.
37. R. K. Gupta and Y. Zorian, Introducing core based system design, *IEEE Des. Test Comput.,* **14** (4): 15–25, Oct–Dec 1997.

LEE A. BELFORE, II
Old Dominion University

AUTOMATIC TESTING FOR INSTRUMENTATION AND MEASUREMENT.

See AUTOMATIC TEST EQUIPMENT.