Automatic Test Generation for Digital Electronic Circuits

Tapan J. Chakraborty
Scott Davidson
Fadi Maamari
Kwang-Ting Cheng

Automatic test generators produce test vectors for a digital circuit, given a description of the circuit at the gate level. For small and testable circuits, a high-quality test can be produced without modifying the circuit. For larger, harder-to-test circuits, the technique of partial scan can provide excellent fault coverage with limited circuit modification. Where test modification can not be tolerated, test-generation tools can help by providing a powerful testability diagnostic capability to assist the designer in writing tests by hand.

Introduction

As mentioned in the introduction to this issue,[1] the need for increasing the quality of AT&T products has made the requirement for high-fault coverage more urgent than ever. Fault coverage is defined as the number of faults detected by a test, divided by the number of possible faults in a circuit. Fault coverage is the accepted measure of test quality. The higher the fault coverage, the less likely it is that defective products will be shipped to customers.

Traditionally, fault coverage was obtained by handwritten tests. As integrated circuits (ICs) have become larger, however, and as competitive pressures have forced development schedules to shrink, handwritten tests have become less and less practical. Therefore, designers have turned to other techniques to obtain tests.

One solution is to create a computer program to write tests, relieving the designer or the test engineer of the task. A program to do this, called an Automatic Test Pattern Generator (ATPG), is the subject of this article. An ATPG assumes that circuits fail in a particular way, called the fault model, and generates tests to detect faults described by this model.

When ATPGs were first employed in testing in the late 1960s and the early 1970s, it was soon discovered that they were very difficult to write. The circuit had to be modified, for increased testability, to achieve acceptable fault coverage. The first ATPGs were used only for combinational circuits, that is, circuits without memory elements. The technique of full scan[2] was invented to make it possible for the ATPGs to be run on real circuits, which almost always include state elements, such as flip-flops and latches (both are types of memory elements). Full scan involves transforming all flip-flops in a circuit into scan flip-flops, thus changing a sequential circuit into an easier-to-test combinational circuit.

While full scan has been used successfully, it has a cost. The transformation of flip-flops into scan flip-flops takes more space on the chip, and, more importantly, decreases the maximum speed at which the circuit can be run. Therefore, research continued to find ATPGs that did not have the limitations of combinational ATPGs, but which could be run on sequential circuits, that is, circuits with memory elements, as well as combinational circuits. The section of this article, "Sequential ATPG," describes AT&T's sequential-test generator, the GENTEST™ test generator, part of the AT&T design-for-testability (DFT) circuit-design product, ATTDFT.

However, many circuits are difficult to test. Even advanced sequential ATPGs often cannot find an appropriate, good-quality test in a reasonable amount of time. When this happens, the first step is to use testability diagnostics, described in the section, "Limitations of ATG," to find the problem. It can then be fixed, inexpensively, either by using manual testability improvements or by partial scan, which selects a subset of flip-flops for scan. Partial scan is described in the following section. In either case, the goal of a test-generation system must be to quickly and effectively produce a test with minimal impact to the circuit design.

The techniques just described ensure good testability after a design is complete. An even better way of ensuring a testable design is to consider testability during the design process, to make sure that the design is testable by design. That is, to ensure that the completed design can be tested without further modification. The section, "In-Process Testability Check," describes "just-in-time testability," which involves improving testability as soon as a testability problem is found in a design.

The test tools described above are oriented to the standard gate-level stuck-at fault model, which assumes that all defects can be modeled by a single circuit node taking on a permanent 0 or 1 value, that is, when the gate-level node of the circuit is permanently forced to, or stuck at, a logical "0" or logical "1" state. The premise of the "stuck-at" fault model is that any product defect that we need to test for may exhibit a "stuck-at" symptom when an appropriate sequence of test inputs is applied. Real defects are much more complex and, as quality demands increase, we are moving to test methods for more sophisticated fault models. The last section, "Future of ATPG," describes advances in automatic test generation that address this issue.

High-fault coverage, and high-defect detection of the ICs that are the heart of AT&T products, will help ensure that we deliver quality to our customers.

Acronyms Used in This Paper

ASIC — Application-specific integrated-circuit
ATPG — Automatic test pattern generator
ATTDFT — AT&T's design-for-testability circuit-design product
BIST — Built-in self-test
DFT — Design for testability
D latch — Simple edge triggers flip-flop with one data (D) input
FF — Flip-flop memory element
IC — Integrated circuit

Sequential ATPG

An ATPG for sequential circuits has to consider several issues that are not present in an ATPG for combinational circuits:

- The much greater "search space" (the number of decision points, and choices at each point) in generating a test for a sequential circuit,
- Complex timing issues, and
- Circuit structures, such as buses, rather than just simple gates.

Test Generation for Multiple Time Frames. Exploring the search space to find a test for a target fault is done through two major steps, *justification* and *implication*. In the justification phase, one of several possible logic-value assignments (that is, a 1 or 0) is made on the input of a node in order to produce a value on the output of the node, which is necessary for the test. The selection of one of several choices of logic-value assignments is also referred to as a *decision* in the search. After each decision, the test generator's implication process is called to identify all the logic-value assignments that are implied by the decision. For example, if a decision assigned a logic value of 1 on the output of an inverting gate, then a value of 0 is implied on the input of the gate.

A conflict appears when both a logic 0 and a logic 1 are implied on the same node by the test generator. When such a conflict occurs, the last decision is "backtracked," that is, all the values implied by it are erased and an alternate choice is made. If the alternate choices have been exhausted, then no test exists for the targeted fault. It is then marked by the test generator as untestable.

Faults in a sequential circuit may require the application of several consecutive input patterns in order to be detected. To find such a test, the ATPG tool has to search for a test through many clock cycles, using time-frame expansion, as described below. Figure 1a shows the general representation of a sequential circuit, consisting of a combinational block with primary inputs, primary outputs, and feedback through memory elements. Figure 1b shows the same circuit expanded in time into multiple time frames, with consecutive time frames representing the state of the circuit under consecutive input patterns. In each time frame, the *inputs* of the memory elements of the circuit are the outputs (called pseudo-primary outputs) of the time frame, and *outputs* of the memory elements of the circuit are pseudo-primary inputs of the time frame. The pseudo-primary output values of one time frame are the pseudo-primary input

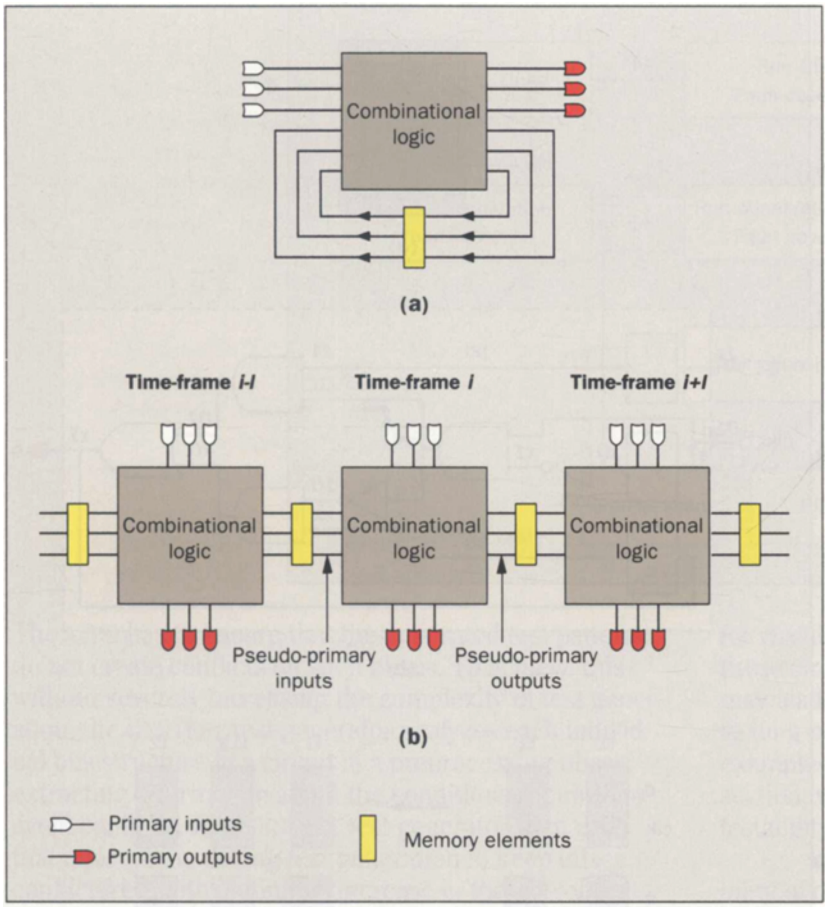


Figure 1. The time-frame expansion of a sequential circuit is shown. In (a), the standard model of a sequential circuit is shown. This model includes three parts: the combinational logic portion of the circuit, the primary inputs and outputs of the circuit, and the memory elements of the circuit, which can be considered pseudo-primary inputs and outputs. In (b), the time-frame expansion model for sequential-test generation is shown. Here, the combinational logic is replicated for each time frame, connected by the memory elements. This indicates the increased complexity of sequential ATPG over combinational ATPG, which deals with only one time frame.

values of the next time frame. This reflects the fact that when a circuit is clocked, the input of a memory element is latched (stored in memory) onto its output.

The search for a test through several time frames can put enormous strain on the complexity of, and memory requirements for, the automatic test generator. In the GENTEST test generator, the Back algorithm[3] is used to make that search more efficient. The algorithm starts the search at primary outputs of the circuit, and goes backwards in time to find a test for the targeted fault. A benefit of the method is that it only needs to work on two time frames at a time. The algorithm is complete—given enough time, it will find a test if one exists.

Clock and Timing Issues. Some circuits have a single, synchronous clock feeding all the memory elements. In such circuits, the clocking can be implicit, that is, the test generator can assume that the circuit will be clocked between time frames, and, therefore, concentrate on finding the appropriate values for the remaining inputs of the circuit. In most circuits, however, multiple clocks are used, and the test generator has to generate the clocking signals explicitly.

Input patterns applied to a sequential circuit will not result in the proper function of the circuit if certain timing constraints are not met. Figure 2a shows a positive-edge, triggered-D latch, which is a type of memory element. When a 0 to 1 transition takes place on its clock input, the value of its data input *(D)* is latched onto its output *(Q)*. For the latch to function properly, the *D* input must be set some time before the clock transition (set-up time) takes place, and must be stable for some time after the clock transition (hold time). The GENTEST test generator addresses the set-up-time and hold-time issues by a combination of modeling and vector (input pattern) expansion.

The gate-level model shown in Figure 2b is used

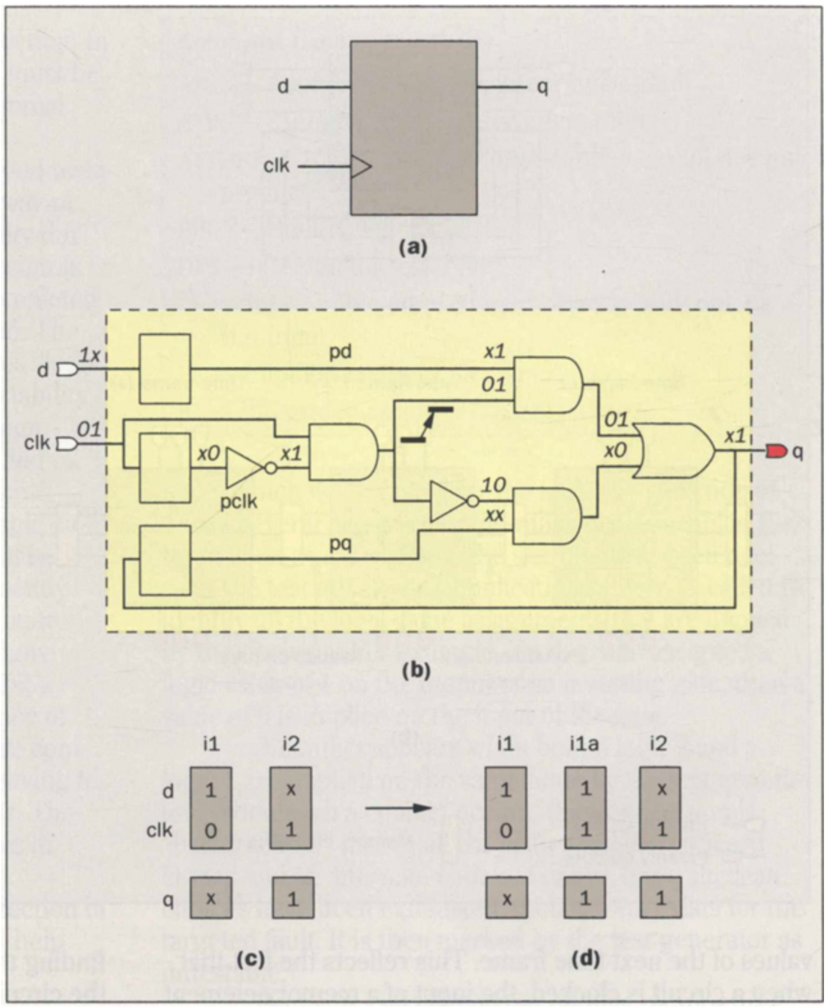


Figure 2: In test generation for a sequential element, (a) shows a flip-flop with data *(d)* input, clock *(clk)* inputs, and data *(q)* output. In (b), the GENTEST test generator model of a flip-flop is shown. Delay elements capture the previous value of the *(d)* input (to model set-up time requirement), clock (to capture the clock edge for edge-triggered flip-flops), and *(q)* output (to preserve the state of the flip-flop). Vectors generated for a flip-flop are shown in (c). If *X* on the *(d)* input arrives before 1 on the clock input, there is a possibility that *X* will be latched into a flip-flop state. In (d), expanded vectors are shown. Vector expansion ensures only one input change at a time, guaranteeing that a 1 is latched into a flip-flop state.

to represent the D latch. It uses three basic memory elements, called *pd, pclk,* and *pq* for the data input, clock, and data output, respectively. These three elements are the values of *d, clk,* and *q*, respectively, in the previous time frame. If *clk* = 1 and *pclk* = 0, a rising transition is detected and *q* = *pd*. Otherwise, *q* = *pq*. During test generation, if a value of 1 is needed on *q*, the two input patterns *i1* and *i2* of Figure 2c are generated. Using preprocessing information that identifies data and clock inputs, the GENTEST test generator then expands the original patterns by inserting an intermediate pattern *i1a*, which is formed by combining the data values of *i1* and the clock values of *i2*. The resulting three patterns achieve the desired output, while respecting both the set-up and hold-time constraints of the latch.

Brief, temporary signal changes, such as hazards and glitches (the brief change of a signal on a circuit), can invalidate a test for a sequential circuit if they reach asynchronous control lines, such as clocks, and asynchronous resets and presets (which change the value of a signal in memory). To prevent this, the GENTEST test generator identifies, in a preprocessing phase, the logic that drives asynchronous controls and marks it as critical. The GENTEST test generator then puts additional constraints on the critical logic, during a test generation, to ensure the absence of test-invalidating hazards.

Complex Circuit Structures. Sequential circuits frequently contain buses that can be driven by multiple drivers that are usually tristate gates. These are drivers that have three output states, 1, 0, and high impedance.

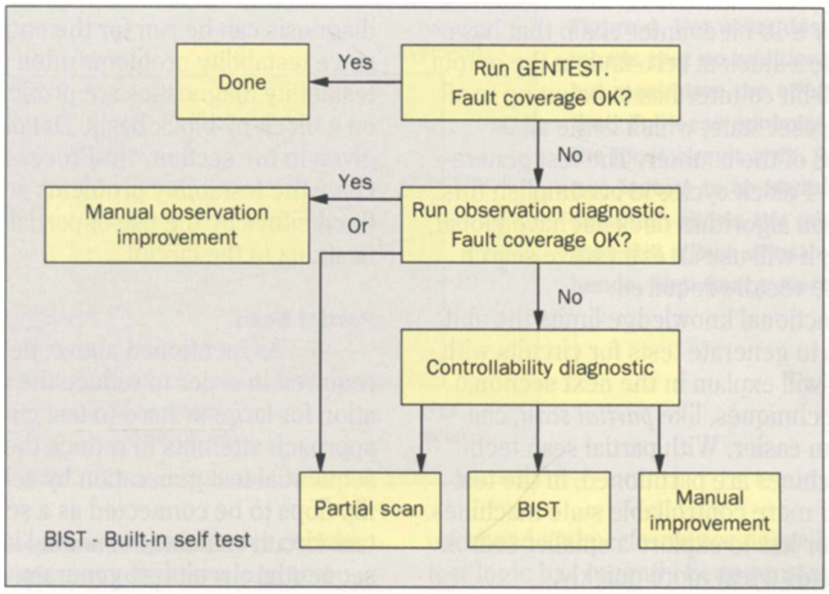

Done
Yes
Run GENTEST. Fault coverage OK?
No
Yes
Manual observation improvement
Or
Run observation diagnostic. Fault coverage OK?
No
Controllability diagnostic
Partial scan
BIST
Manual improvement
BIST - Built-in self test

Figure 3. For each circuit and circuit block, the GENTEST test generator diagnostic features can indicate if the design is testable and, if not, if the problem is poor controllability or poor observability. If there is a problem, either manual or automatic (built-in self-test or scan) techniques can be used to create a testable design.

The ATPG has to ensure that the generated test patterns do not create conflicts on such buses. To achieve this without severely increasing the complexity of test generation, the GENTEST test generator analyzes each individual bus structure in a circuit in a preprocessing phase, extracting information about the conditions required to avoid conflicts. The GENTEST test generator then uses that information during test generation to keep buses conflict-free, with a minimal increase in the size of the test-generation search space.

Circuits with large memories, such as random access memories (RAMs), pose another challenge. While a test for the internals of the RAM is best accomplished using built-in self-test techniques, the ATPG must be able to justify faults (that is, find the input value of a circuit) of the RAM needed for the test, and do implication (that is, drive the value through the RAM). A memory model must be built into the test generator. Special memory-management techniques must be used to store the faulty state of a circuit using RAM. This is so because a small 16-kilobit RAM in a circuit with 20,000 faults could require up to 320 megabytes of memory, if one uses a naive modeling technique, that is, copying the values of the whole memory for each faulty machine.

Limitations of Automatic Test Generation

Automatic test generation (ATG) for sequential circuits can be very difficult, and sometimes impossible, for circuits with poor testability. Certain structures in these circuits, which are required to ensure functionality, may lead to poor testability. Large-state machines, such as long counters and internal-clock generators, are some examples of circuit structures with poor testability. In this section, we describe a diagnostic method to identify these testability problems by using the GENTEST test generator.

Circuit Structures Causing Poor Testability. The testability of digital circuits is evaluated with two parameters, *controllability* and *observability*. The controllability of a node in a digital circuit is the measure of how easily a circuit node can be set to a desired logic value with vectors applied at the primary inputs. Observability is the measure of how easily the logic value at a circuit node can be propagated to a primary output for observation. Controllability and observability, getting the value to an output, determine the testability of a circuit node. To generate a test for a fault deep inside a sequential circuit, a number of vectors have to be generated by the ATPG, first to activate the fault, and then to propagate the fault effect to a primary output. If the controllability or observability of a circuit node at the fault site is poor, the ATPG may have to generate a large number of vectors. This requires a large amount of memory and time, and may not be practical at all for very large or very untestable designs.

Large-state machines, for instance, a long counter, are examples of circuit structures that are hard to control. Let us consider an example in which a 10-

input AND gate is fed by a 10-bit counter chain that has a reset signal. To activate a stuck-at zero fault at the output of this AND gate, the 10-bit counter has to count up to all the 1s states from the reset state, which is the all 0s states at all the flipflops of the counter. The test generator has to generate 2^{10}-1 clock cycles to accomplish this. Since the test generation algorithm does not have global knowledge of a circuit, it will use an exhaustive search process to generate the vectors required.

The lack of functional knowledge limits the ability of sequential ATPGs to generate tests for circuits with poor testability. As we will explain in the next section, design-for-testability techniques, like *partial scan*, can make the ATPG problem easier. With partial scan techniques, large-state machines are partitioned, in the test mode, into smaller and more controllable state machines. Thus, the test generator has to explore a smaller search space, and therefore finds a test more quickly.

Similarly, if some circuit structure embedded in a sequential circuit is driven by an internally generated clock, the test generator has to activate the internal-clock generation subcircuit for each clock period to compute each vector. This is repeated for each test vector for the entire vector set, leading to a long test-generation time. But if this internally generated system clock is multiplexed with another clock signal from the primary inputs, then, in test mode, the test generator can activate the more controllable alternate-clock signal and generate test vectors more quickly.

Testability Diagnosis. The GENTEST test generator can be used to diagnose these testability problems. In diagnosis mode, the cause of the untestability of a fault is identified, particularly whether the cause is poor controllability and/or observability. The diagnosis process is shown in Figure 3. First, a circuit is run through the GENTEST test generator. If a test is generated, then the circuit is testable. If not, the faults not detected are noted in a file. The GENTEST test generator can be run in the observability diagnosis mode with these undetected faults. In this mode, all the circuit nodes are considered observable, and if some undetected faults are now detected, then it can be concluded that the testability problem for those faults is poor observability. The faults that are still undetected in the observability mode suffer from poor controllability.

Thus, a sequential-circuit test generator can be used for diagnosing testability problems in a circuit. This diagnosis can be run for the entire circuit. However, since testability problems often appear within a block, testability diagnostics are profitably run during design, on a block-by-block basis. Details of this process are given in the section, "In-Process Testability Check." Once the testability problems are identified, they can be fixed either by the use of partial scan or by manual modifications to the circuit.

Partial Scan

As mentioned above, design for testability is still required in order to reduce the complexity of test generation for large or hard-to-test circuits. The partial scan approach attempts to reduce the complexity of sequential-test generation by selecting a subset of circuit flip-flops to be connected as a scan register. The resultant circuit remains sequential in the test mode, and a sequential-circuit test generator is used. In this section, we present the partial scan, flip-flop selection methods used in the GENTEST test generator.

Cycle-Breaking. Why is it sometimes difficult to generate tests for sequential circuits? The answer is that the cycles in the circuit graph, where nodes are the flip-flops, or memory elements, and the edges are the combinational signal dependencies, are mainly responsible for test generation complexity. (In graph theory, edges are the paths between vertices, which in this case are the flip-flop memory elements.) In the absence of such cycles, even a large sequential depth, that is, the number of flip-flops between input and output, causes no serious problem to the test generator. Also, cycles of length greater than one pose a more serious problem to test generation than do cycles of length of one (self-loops). Therefore, the first partial scan flip-flop selection criterion is to break all cycles of length greater than one by scanning a minimal subset of flip-flops.[4,5]

Self-loops are not broken at the cycle-breaking stage to keep the scan overhead low, since the number of self-loops in most circuits is quite large. The algorithm used to break global cycles was presented in Bhawmik et al.[5] The algorithm first detects, and then eliminates, all strongly connected components in the directed-circuit graph in a hierarchical manner. Strongly connected components are eliminated by recursively deleting nodes. This is done by making them scan flip-flops until the entire graph becomes acyclic, that is, the edges no longer form a cycle.

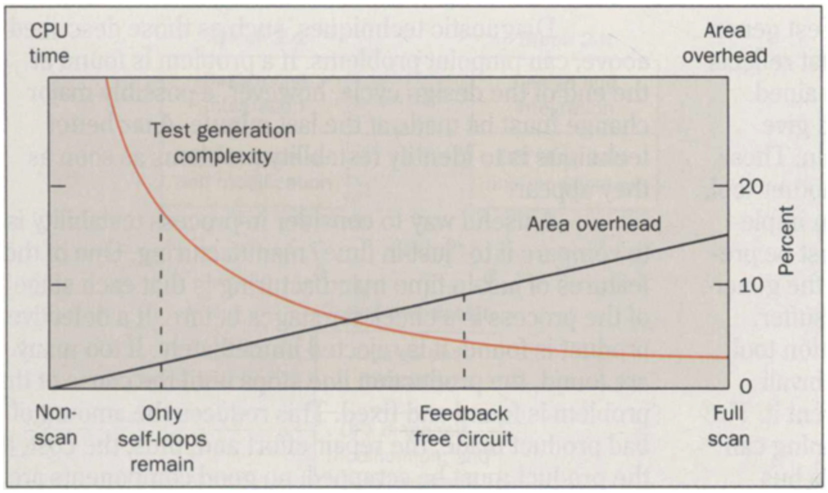


Figure 4. Not using design for testability (DFT) means that no additional circuit area is needed, but maximizes the effort required to generate a test. Full scan minimizes this effort, but requires the most circuit area. Partial scan allows these two factors to be traded off. Effective scan selection allows the minimum area overhead that still allows effective test generation and, hence, high fault coverage.

Reduction of Self-Loop Path Length. Under certain circumstances, the self-loops may result in a long test sequence. Some implementations of a counter contain only self-loops, for example, and the number of vectors needed to detect some faults in the counter is exponential. For large circuits without long cycles, the test generation complexity may still be high if it contains long, *consecutive self-loops*. The *distance* along a directed path between two vertices is defined as the number of vertices on that path.

For a graph that has no cycles of length greater than one, we say the graph has a consecutive self-loop of length *K* if there exists a directed path of distance *K*, and every vertex along the path has a self-loop. To reduce the test-generation complexity of large circuits to a manageable level, we further select flip-flops, to reduce the length of the longest consecutive self-loop, after breaking all global cycles.

Figure 4 shows the tradeoff curve of area-overhead vs. test-generation complexity. For medium-size circuits, a test generator could generate high-coverage tests for partial scan circuits without cycles of length greater than one. For very large circuits, some of the self-loops in a long chain of consecutive self-loops need to be broken to reduce test-generation complexity to a manageable level.

Timing-Driven Partial Scan. Besides low-area overhead, one key advantage that partial scan design has over other DFT techniques is low, or zero, performance degradation if the scan flip-flops are selected carefully. A partial scan approach that aims at minimizing both area overhead and performance degradation caused by the test logic has recently been proposed.[6] Because meeting performance objectives is one of the most difficult and time-consuming tasks in high-performance, application-specific integrated-circuit (ASIC) design, the selection of scan flip-flops should be guided to meet the given performance requirement.

If the minimum scan flip-flop selection for adequate testability still causes the circuit's performance requirements not to be met, a performance optimization will have to be done. The selection of scan flip-flops should be guided to reduce the potential extra area caused by the performance optimization. Given a target speed and an initial design that meets the target, the algorithm proposed in Jou at al.[6] selects a minimum set of scan flip-flops, if they exist, that:

- Will break all sequential cycles, and
- Will not violate the performance requirement after the scan logic is added.

If such a set does not exist, the algorithm will find such a set of scan flip-flops in which:

- All sequential cycles are broken, and
- The total area increase caused by the scan logic and the subsequent performance optimization is minimized.

The algorithm uses timing-analysis data to guide the flip-flop selection process. It avoids the selection of flip-flops on critical paths. Experimental results show that, for most circuits, cycles can be broken without degrading the performance of the circuit.

Inserting Partial Scan. Once the partial scan flip-flops are selected, the circuit must be modified to insert the scan chain. This is not enough, however. Test generation must be done on a model of the circuit that reflects the greater controllability and observability obtained through the use of the scan. Vectors generated give explicit values for each flip-flop in the scan chain. These can be sequenced into the actual test, using another tool, after test generation is complete. In partial scan implementations, the states of non-scan flip-flops must be preserved during the scan-in process. Otherwise, the generated test will be invalid and fault coverage will suffer.

To guarantee this, the testability insertion tool must identify situations in which states can be invalidated, and insert appropriate hardware to prevent it. The tool also must identify situations in which scanning can cause illegal circuit states—which might lead to bus conflicts, for instance—and again insert logic to prevent the problem. For circuits with complex, asynchronous control structures, which are quite common in telecommunications products, there may be hundreds of corrections to be made. It is vital that these corrections be done automatically to:

- Avoid the possibility of error introduced by manual modification,
- Reduce test insertion time, and
- Make testability insertion a truly push-button process.

The combination of effective flip-flop selection and automatic circuit modification produces a robust, effective solution to the testability problem.

In-Process Testability Check

We have seen how testability diagnostics can be used to find and eliminate testability problems, and how partial scan can be used to automatically create a testable circuit. In this section, we describe how these techniques fit into the design process.

How do the hard-to-test structures described in the earlier section, "Limitations of Automatic Test Generation," get into the circuit? The answer is that they are designed in. Many logic structures required to implement the functionality of telecommunications circuits are fundamentally hard to test. So, testability problems are not mistakes, they may be side effects of a good design.

The solutions to most testability problems are well understood as soon as the problem is identified. Counters can be split; clocks can be multiplexed. The difficulty is in identifying the problem.

Diagnostic techniques, such as those described above, can pinpoint problems. If a problem is found at the end of the design cycle, however, a possibly major change must be made at the last minute. A far better technique is to identify testability problems as soon as they appear.

A useful way to consider in-process testability is to compare it to "just-in time" manufacturing. One of the features of just-in-time manufacturing is that each stage of the process is a check for stages before. If a defective product is found, it is rejected immediately. If too many are found, the production line stops until the cause of the problem is found and fixed. This reduces the amount of bad product made, the repair effort and, thus, the cost. If the product must be scrapped, no good components are added to the bad ones.

In-process testability, which we call "just-in time testability," has similar advantages. By finding testability problems early, the designer can fix them early. This means that the problem does not have to be diagnosed and fixed for the complete design. Just after the design of a block of the circuit is done, the designer knows the structure far better than when the entire design is complete and, thus, can find the problem faster. Also, the designer will not make the same mistake again. This means that the problem will be fixed once, not many times throughout the circuit. Finally, time pressures are not as great during design as at the end of the process, so the problems can be fixed without the pressures that often lead to mistakes.

Figure 5 shows how just-in-time testability works. When the design of the smallest block in the circuit is complete, a test is generated using the sequential ATPG. If the block is reasonably testable, this should take no more than a few minutes. If high-fault coverage is achieved, it means that this block is testable, at least by itself. The generated vectors are discarded, since they will not be useful in the chip as a whole. But the vectors were automatically generated, so not much time and effort is lost.

If the block does not achieve a high-fault coverage, or test generation takes a long time, the block has a testability problem. Any difficulty in generating a test for the block by itself will be magnified many times when the block is embedded in the chip. The testability diagnostics for automatic test generation, described in the section, "Limitations of Automatic Test Generation," can

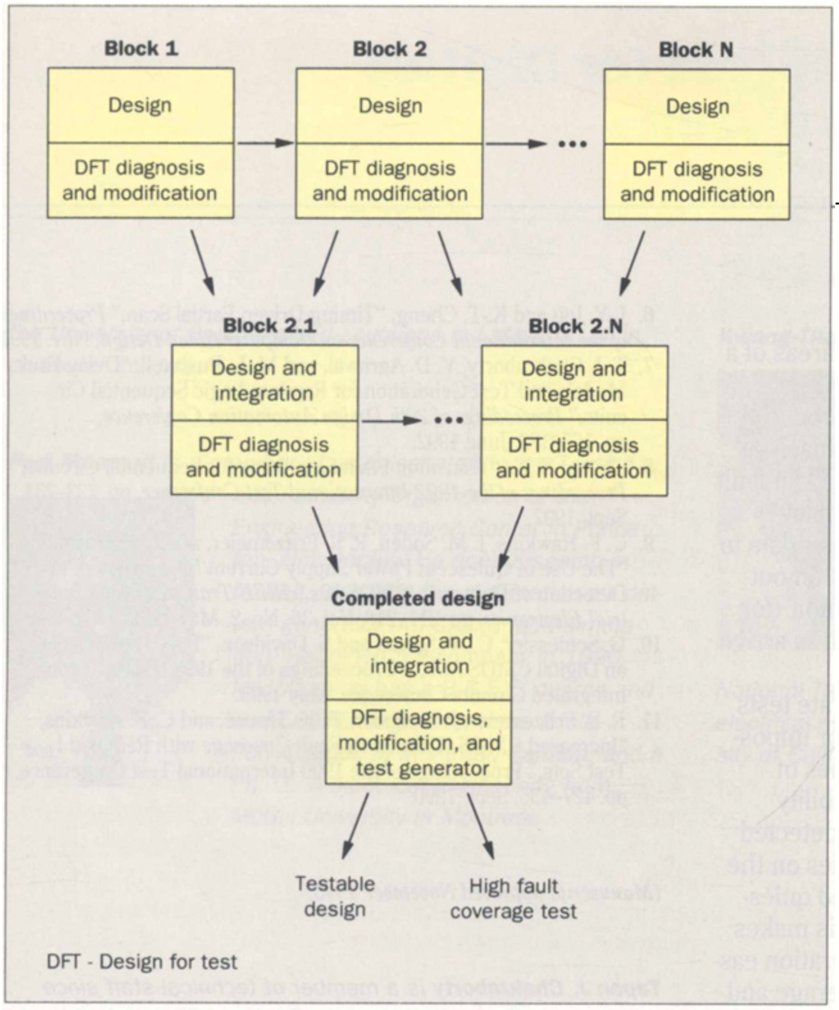


Figure 5. With "just-in-time testability," each module of a design is checked for testability as soon as the design is complete. The testability of hard-to-test modules should be enhanced immediately. Testable blocks are combined into larger blocks and rechecked, until the entire design is complete.

be used to find the problem. The block can either be redesigned, or the flip-flops, which have been selected for partial scan by the controllability diagnostic, can be saved for a partial scan implementation.

Once all the blocks at one level of the hierarchy are considered, some are combined into a block at the next level. The combination of testable blocks is not necessarily testable. This process continues until the entire chip is considered.

There are three outcomes at the chip level:

- The changes to the blocks have made the chip testable as a whole, and a test can be generated for faults remaining after the functional vectors have been fault-simulated.
- Scan flip-flops have been selected for hard-to-test blocks. A full-chip, partial scan insertion is done using these flip-flops, and a test is generated. Often, fewer scan flip-flops are selected by working up from the block level than would be chosen if the selection were done for the entire design.
- Because of the size of the design, test generation cannot achieve adequate fault coverage, and cost or performance requirements make it impossible to use a scan. In this case, the chance of achieving high fault coverage through manually written vectors is much higher, because the structures that cause low testability have already been repaired.

A final advantage of in-process testability is that it points out, early on, a possible need for structured test solutions. All structured solutions, whether scan or built-in self-test (BIST), work better for certain design styles. If the need for these is found early, the design can be done for easy scannability, or "BIST-ability." This saves time at the end of the design cycle and produces a chip that is easier to test.

The Future of Automatic Test Generation

The algorithms and tools described above provide a good solution to the problem of getting high stuck-at fault coverage for digital designs, given the freedom to modify the circuit to increase testability. But this is still not the ultimate answer. What is missing?

At present, test generators create tests for stuck-at zero and one faults. But these faults are not really the way in which circuits fail. Circuit failures actually are caused by physical defects:

- Open defects, in which a lead has a break,
- Bridging defects, in which two wires or two areas of a transistor are shorted together, or
- More subtle failures, such as gate oxide shorts.

These defects sometimes manifest themselves as hard failures, which can be detected by a stuck-at fault test, and sometimes as reliability problems or delay faults. A delay fault is a fault that causes incorrect data to be latched into a memory element, or appear at an output, because of a slow path that causes a transition (for example, a change from 0 to 1) on the data path to arrive after a clock transition.

It is possible, though difficult, to generate tests for delay faults.[7,8] However, it is very difficult or impossible to generate tests to detect many other types of defects, especially those causing potential reliability problems. Many of these other defects can be detected by a technique called I_{DDQ},[9] testing, which relies on the fact that defective CMOS circuits exhibit elevated quiescent current draw, which can be measured. This makes faults more observable, which makes test generation easier,[10,11] and moves testing away from fault coverage and toward defect coverage.

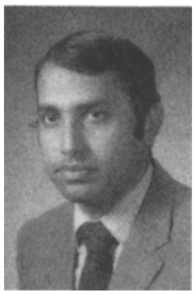
Conclusion

Automatic test generation offers several benefits for today's designs. For small or testable designs, sequential ATPGs can generate a high fault-coverage test much more quickly than is possible by hand. Where this is not possible, it can be used to find testability problems in a design and, when used in the design process, help reduce the increases in design time caused by poor testability. Finally, when combined with partial scan, sequential ATPG can provide an effective test solution at very little cost.

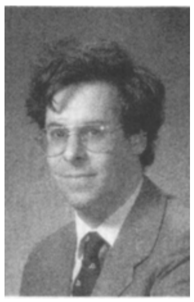
References

1. R. L. Campbell and R. A. Tarbox, "Testing Goes Critical Path," *AT&T Technical Journal*, Vol. 73, No. 2, pp. 4–9.
2. T. W. Williams and K. P. Parker, "Design for Testability - A Survey," *Proceedings of the IEEE*, Vol. 71, No. 1, Jan. 1983, pp. 98–112.
3. W.-T. Cheng, "The Back algorithm for sequential test generation," *International Conference on Computer Design*, pp. 66–69, October 1988.
4. K.-T. Cheng, V. D. Agrawal, "A Partial Scan Method for Sequential Circuits with Feedback," *IEEE Transactions on Computers*, pp. 544–548, April 1990.
5. S. Bhawmik, C. J. Lin, K.-T. Cheng, V. D. Agrawal, "Pascant: A Partial Scan and Test Generation System," *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1991.
6. J.-Y. Jou and K.-T. Cheng, "Timing-Driven Partial Scan," *Proceedings of the International Conference on Computer-Aided Design*, Nov. 1991.
7. T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell, "Delay Fault Models and Test Generation for Random Logic Sequential Circuits," *Proceedings of 29th Design Automation Conference*, pp. 165–172, June 1992.
8. Cheng, K. T., "Transition Fault Simulation for Sequential Circuits," *Proceedings of the 1992 International Test Conference*, pp. 723–731, Sept. 1992.
9. C. F. Hawkins, J. M. Soden, R. R. Fritzemeier, and L. K. Horning, "The Use of Quiescent Power Supply Current Measurement in Detection of Defects in CMOS ICs," *IEEE Transactions on Industrial Electronics*, pp. 211–218, Vol. 36, No. 2, May 1989.
10. G. Schiessler, C. W. Spivak and S. Davidson, "I_{DDQ} Test Results on Digital CMOS ASIC," Proceedings of the 1993 IEEE Custom Integrated Circuits Conference, May 1993.
11. R. R. Fritzemeier, J. M. Soden, R. K. Treece, and C. F. Hawkins, "Increased CMOS IC Stuck-At Fault Coverage with Reduced I_{DDQ} Test Sets," Proceedings of the 1990 International Test Conference, pp. 427–435, Sept. 1990.

(Manuscript approved November 1993)



Tapan J. Chakraborty *is a member of technical staff since 1986 at the Test and Diagnosis Department of AT&T Bell Laboratories in Princeton New Jersey. He is working in the area of delay fault testing and timing analysis for very large scale integrated (VLSI) circuits. Mr. Chakraborty has a B.S.E.E. degree and an M.S. degree in computer science from Jadavpur University, India, and an M.S. degree and a professional degree in Electrical Engineering from Columbia University, New York City, and a Ph.D. in computer engineering from Rutgers University in New Brunswick, New Jersey.*



Scott Davidson *is a technical manager of the Design for Testability and Test Data Management Group at the Engineering Research Center in Princeton, New Jersey. The group is responsible for developing digital test algorithms and tools. Mr. Davidson joined AT&T in 1980. He has a B.S. degree from the Massachusetts Institute of Technology in Cambridge, an M.S. degree from the University of Illinois, Urbana-Champaign, and a Ph.D. from* *the University of Southwestern Louisiana in Lafayette, all in computer science.*

***Fadi Maamari** is a member of technical staff in the Testing and Reliability Organization of the Engineering Research Center in Princeton, New Jersey. He does research in testing, including the automatic test pattern generator (ATPG) and fault simulation. He joined the company in 1990. Mr. Maamari has a B.S.E.E. degree and an M.S.E.E. degree from the Ecole Polytechique in Montreal, Canada, and a Ph. D. in electrical engineering from McGill University in Montreal.*

***Kwang-Ting (Tim) Cheng** recently joined the faculty at the University of California at Santa Barbara as an associate professor of electrical and computer engineering. Previously, he worked at AT&T Bell Laboratories in Murray Hill, New Jersey, as a member of technical staff. His research area is computer-aided design for very-large-scale integrated circuitry, with a special focus on testing and synthesis. He received his B.S.E.E. degree from the National Taiwan University, Taipei, Taiwan, and a Ph. D. in electrical engineering and computer science from the University of California at Berkeley.*