Testing ICs: Getting to the Core of the Problem

Brian T. Murray
General Motors R&D Center

John P. Hayes
University of Michigan

This tutorial examines the market and technology trends affecting the testing of integrated circuits, with emphasis on the role of predesigned components—cores—and built-in self-test

Today's integrated circuits are larger, faster, denser, and more power-hungry than ever before. These trends have made testing ICs more difficult and expensive. Although IC design and manufacturing methods are obviously important, the production of reliable, high-quality ICs requires high-quality testing.

Larger chips require more tests and are difficult to test efficiently. Faster chips can't always be tested with current tools. Chips that are densely packed with features are more easily affected by a single defect and susceptible to new failures that traditional techniques might not detect. And power-hungry chips have caused designers to reduce voltage ranges, which also makes chips more susceptible to minor defects.

Marketing issues also influence testing. Cost, of course, is a key factor, as is time-to-market. In today's market, design time is critical to profitability. If adding special circuits will make a chip easier to test, it could speed delivery and so be worth the additional cost. On the other hand, the additional circuits can also affect IC package size as well as how many working chips can be manufactured in a single batch, all of which increase cost. Also, automated test equipment is expensive—the purchase price of ATE can offset the benefit of a small increase in market share.

For example, Table 1 shows the economics of Intel's Pentium microprocessor. Just a 1-percent increase in the size of a Pentium increases its annual production cost by $63.5 million. A 15-percent increase in size not only costs $961 million but reduces the number of chips that can be produced, requiring a new fabrication plant to maintain production levels.[1]

Table 1. Estimated effect of die size increases on Pentium fabrication costs.

	Nominal Pentium die	1-percent die size increase	15-percent die size increase
Wafer cost	$1,460	$1,460	$1,460
Die size	160.2mm^2	161.8mm^2	184.2mm^2
Die cost	$84.06	$85.33	$102.55
Added annual cost	—	$63.5M	$961M
Dies required/week	1M	1M	1M
Chips fabricated/week	498.1K	482.9K	337.5K

Originally published in *Computer*, Nov. 1996, pp. 33–38.

Copyright © 1996 IEEE, Inc. All rights reserved.

To save time and money and to improve reliability, designers are increasingly reusing large, well-tried standard components, which have recently come to be called *cores*. Widespread use of cores is a new phenomenon and it creates some challenging testing problems. One method of testing core-based designs that is rapidly becoming popular is *built-in self-test*. In BIST the special circuits that produce the tests needed by the cores are included on chip, as are other circuits that monitor and evaluate the test responses.

This tutorial examines the market and technology trends affecting the testing of integrated circuits, with emphasis on the role of cores and BIST. Here we explain manufacturing testing, as opposed to design testing, which happens before manufacturing, and on-line testing, which happens after.

Testing Basics

A system *fails* when the service it provides differs from the service it was designed to provide. A system is in *error* when its state differs from the correct state. A *fault* is the physical difference between a good system and a failing one.

Faults in chips arise during processing, either from defects in the semiconductor wafer or from minor variations in processing due to impurities or vibrations. The goal of testing is to detect faults and thus prevent failures.

Figure 1 outlines a generic test procedure. The test-generation step in the middle of the figure has two basic parts:

1. Activate, or expose, the selected fault.
2. Propagate an error signal from the fault site to an observable output, such as the pins of a chip.

Testing combinational circuits

Figure 2 shows a sample combinational circuit, the smallest member of a widely used set of benchmark circuits. Faults are represented by maintaining a single logic line in the circuit at a constant value 0 or 1 regardless of how the circuit stimulates the line. For instance, to test if a line is stuck at 0, the test must find a sequence of one or more input patterns that will make the targeted line 1.

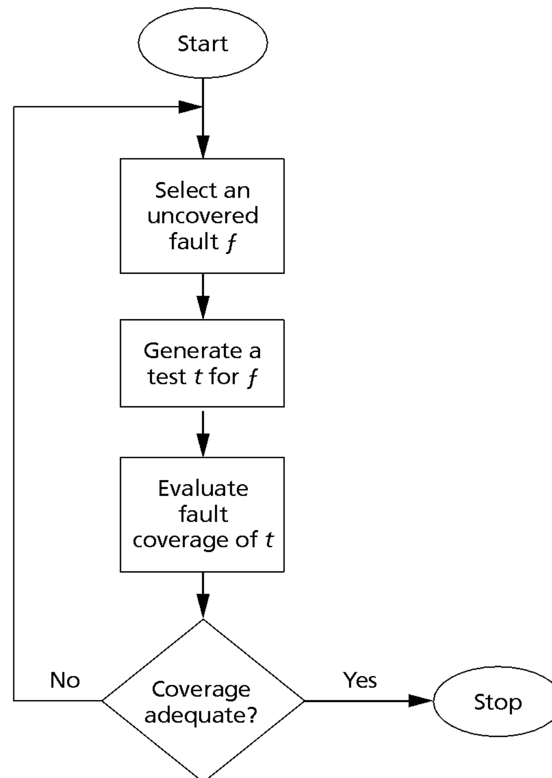

Figure 1. A generic test generation procedure.

Copyright © 1996 IEEE, Inc. All rights reserved.

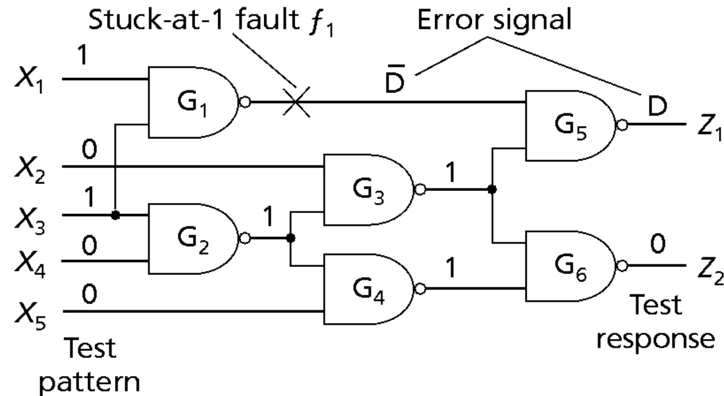

Figure 2. Sample test generation. The figure uses standard notation: If a line in a circuit is 0 (1) when it should be 1 (0), the error signal value on that line is represented by D ($\overline{D}$) for discrepancy.

In Figure 2, a stuck-at-1 fault, f_1, at the output of NAND gate G_1 can be exposed by forcing G_1's output signal to 0. G_1's output will be in error if it is 1 when it should be 0. The fault can be exposed, therefore, by assigning 1 to both X_1 and X_3. To observe fault f_1, the test must propagate the error signal (denoted D or its inverse $\overline{D}$) from G_1 through G_5 to the observable output line Z_1. This process is called *D-propagation*. In this case, G_5 must be *sensitized* to the error signal on its upper input line; that is, a change on this line must cause a change on Z_1. We can achieve this only by assigning 1 to G_5's other input. The process of determining signal values that must occur to meet a desired testing objective is called *implication*.

Now G_3's output has been assigned the logic value 1, but none of its inputs have yet been assigned. Setting X_2 input to 0 fully specifies G_3's desired behavior. The process of determining complete and consistent values at primary inputs that imply desired signal values at internal nodes is called *justification*.

All test-generation algorithms must perform these operations in some fashion.

The Podem (Path Oriented Decision Making) algorithm is the basis for most modern test-generation programs.[2–4] Most new test-generation techniques, including Socrates,[3] extend Podem in an attempt to reduce test-generation time. The goal of CompacTest,[4] on the other hand, is to reduce test set size, which increases tester throughput.

Sequential circuits

It is far more difficult to generate tests for sequential circuits, which contain storage devices such as flip-flops, than for combinational circuits, which do not.

Algorithms such as Podem and its derivatives cannot directly generate tests for sequential circuits because they assume that all assignments are instantaneously propagated. However, they can be extended to generate tests for circuits with sequential behavior. Figure 3a shows the general form of a sequential circuit. For test-generation purposes, a pseudocombinational iterative model, shown in Figure 3b, is derived as follows:

1. Replace each flip-flop, which has a fixed, clock-determined delay, with a pseudo flip-flop whose function is equivalent but whose output is produced instantaneously.
2. Connect multiple copies of the circuit to form a combinational circuit in the iterative form, as in Figure 3b.

Each copy of the iterative circuit, called a *time frame*, represents a different instant in time.

Given a known initial state $q(t_0)$ at time t_0, the most common approach to sequential test generation is to construct an iterative model like this with r time frames. If the test generator cannot generate a test for the circuit in i time frames, it adds a time frame and begins test generation again.[2]

Generating a sequential test may require the construction of many time frames. While most current combinational test generators can generate tests for all combinational circuits in reasonable time, sequential test generators cannot generate tests for all faults in typical sequential circuits.

Originally published in *Computer*, Nov. 1996, pp. 33–38.

Copyright © 1996 IEEE, Inc. All rights reserved.

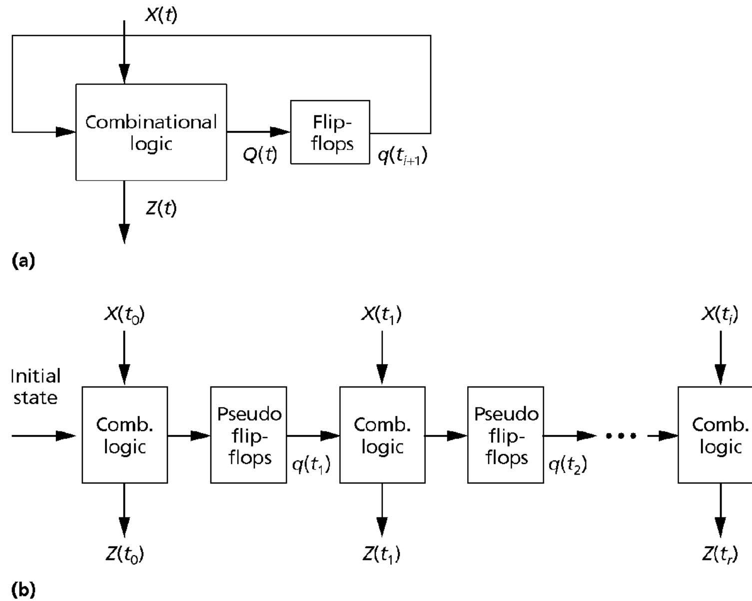

Figure 3. (a) A sequential circuit and (b) an equivalent pseudocombinational model.

It is not unusual for sequential test generators to be unable to produce a circuit state required for a test-generation step when an easy solution is known to the designer. Some test generators, such as Hitest,[5] were developed specifically to incorporate the designer's knowledge about the circuit. In other cases, test-generation performance can be improved by viewing the circuit at a higher level of abstraction. For instance, PathPlan can propagate entire sequences of tests along multibit buses.[2]

Fault models

To develop tests, engineers target fault models—abstract models of expected defects. The most popular fault model is the *single stuck-line*, or SSL model shown in Figure 2. Coverage of SSL faults is currently the only well-accepted measure of test quality. A considerable amount of work has been devoted to verifying the SSL model and to developing new, more accurate gate-level models such as delay faults[2] and bridging faults.[6]

In principle, it is possible to generate tests without using a fault model. One such approach is exhaustive testing, which applies all possible patterns to a circuit. *Exhaustive* testing of entire circuits is not practical because it takes too long. More practical is *random* testing, in which pseudorandomly generated patterns are applied to the circuit inputs. However, the number of pseudorandom patterns required to ensure high fault coverage is sometimes excessive. Although exhaustive and pseudorandom methods are impractical for testing entire chips, they can be used to test small cores.

Test coverage

As noted above, it is desirable to maximize *fault coverage*, or the percentage of potential faults the tests can detect. Many high-volume IC manufacturers believe that SSL fault coverage of at least 99 percent is necessary. Of course, it is also desirable to minimize costs, including the costs of generating and applying tests.

A *fault simulator* models faulty versions of a circuit as well as an unfaulted "good" version. If there are *N* faults to be considered, the fault simulator simulates different circuit responses to a given input in one simulation pass. The outputs from the simulation pass due to the *N* faulty circuits are compared with the one good circuit, and faults associated with incorrect circuit outputs are identified.

To improve the efficiency of test-generation algorithms, fault simulation is used to identify the faults serendipitously covered by each test pattern. Fault simulation must be used to evaluate the coverage of test patterns generated by pseudorandom methods. The process of analyzing coverage via a fault simulator is called *fault grading*.

Pattern depth

Finally, the capabilities of automated test equipment are important. ATE for manufacturing store test pat-

Originally published in *Computer*, Nov. 1996, pp. 33–38.

Copyright © 1996 IEEE, Inc. All rights reserved.

terns in special memories so that they can be automatically applied to the chip. The number of test patterns that can be stored in this way is called the ATE's *pattern depth*. Testers with low pattern depth tend to be slower, but large pattern depth can be uneconomical. Thus, circuits requiring long test sequences can be very expensive to test.

Design for Testability

A circuit can be difficult to test for a variety of reasons. A design process or philosophy that leads to good testability is *design for testability*.

Several well-known design techniques facilitate testability. For example, avoiding logical redundancy, providing for direct initialization of memory devices, and providing a mechanism for logically breaking global feedback loops all make test generation easier and improve fault coverage. Many companies compile long lists of such rules for their designers. However, this ad hoc approach adds considerably to the designer's burden and may still not provide adequate testability.

Another approach is systematic design-for-testability, usually applied and enforced by computer-aided-design tools and now often called *test synthesis*.

Scan design

The basic idea of the most common test synthesis technique, *scan design*[2] is the separation of memory modules from combinational modules during testing. Memory modules, such as flip-flops, are chained together into a shift register called a *scan chain* when a special test mode is activated. As Figure 4 shows, this partitions the circuit into a set of combinational subcircuits C_i, whose inputs and outputs are connected to the scan chain.

Each combinational subcircuit can then be tested by shifting its input test data into the scan chain through the scan input X_s. The data ripples through the logic in parallel, and the results are loaded in parallel into another part of the scan chain. Finally, the data is scanned out serially through Z_s. The scan chain itself can be tested by shifting a pattern of 1s to fill the chain, followed by the same number of 0s, and then the sequence 0101.... Tests can be derived for the combinational subcircuits using any combinational test generator. Connecting all the memory modules of a circuit into a scan chain—*full scan*—obviates the need for explicit sequential test generation.

There are several variations on scan design,[2] each providing controllability and observability of a sequential state, but all requiring strict adherence to special design rules. Today, circuits can be automatically analyzed for design rule violations, scan design can be automatically implemented or inserted into an existing design, and tests can be automatically generated by a number of commercial CAD tools.[7]

Scan design can also be used for purposes other than avoiding sequential test generation. For example, scannable transparent latches can be added to the pins of a chip (*boundary scan*) to provide test-mode access to the interconnections between chips or to access test modes within the chip. A modification of the same technique can be used to test cores within a single chip.

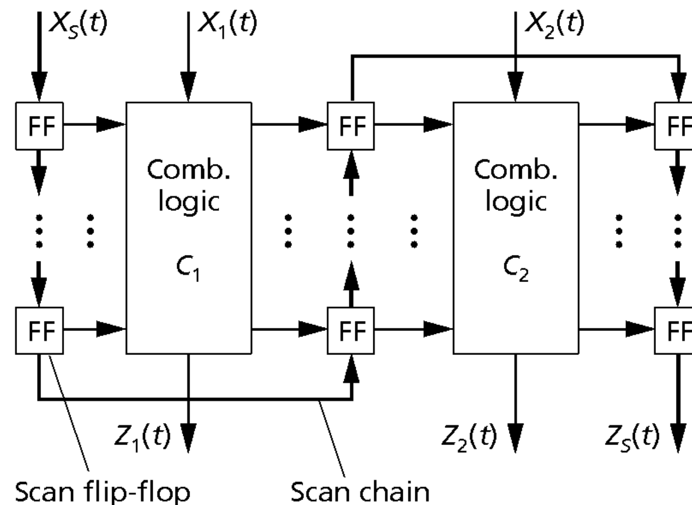

Figure 4. Scan testing scheme.

Copyright © 1996 IEEE, Inc. All rights reserved.

Scan design drawbacks

If scan is so successful, why is there still so much interest in sequential test generation? There are two main drawbacks to scan-based test synthesis. First, scan design rules often prohibit practices that are widely used in the design community. For example, while scan design proponents may condemn the gating of clocks as poor design practice, this technique is often used in high-performance circuits. Scan chains, on the other hand, create longer critical paths that can reduce clock speed to unacceptable levels.

Scan also needs more chip area, which can be unacceptable in cost-sensitive designs. In addition, for circuits with many memory elements, the time it takes to scan test data into and out of the chip serially can be prohibitive, and require expensive ATE with large pattern depth.

Finally, scan design cannot be used at all for dense memory modules such as RAMs. In all the above cases, designers may employ *partial scan*, in which only some memory elements are chained together. Sequential test generators are required for such circuits.

Built-In Self-Test

One design-for-testability technique that has recently gained popularity is built-in self-test. In BIST, circuits are added to the chip to enable it to test itself. To implement BIST, existing structures can be designed to be reconfigurable in test mode, or completely new structures can be added. In both cases, circuit area is increased and performance can be decreased. In addition, like scan design, BIST can introduce undesirable delays in the critical path of a circuit.

However, BIST has several advantages, especially the ability to control and observe deeply embedded cores and test *at-speed* (at normal clocking rates). It reduces the need for expensive external testers—it can even eliminate external testers entirely. Moreover, BIST hardware can be reused during normal circuit operation for on-line testing.

BIST has been effective for a wide variety of circuits: Regular circuits such as RAMs and ROMs, with well-structured sets of test patterns that can be efficiently generated on-chip, are now routinely tested using BIST.

Figure 5 illustrates the basic approach to BIST. When the test signal is not active the circuit behaves normally and receives its inputs from other modules. When the test signal is active, a test-pattern-generator (TPG) circuit applies a sequence of test patterns to the circuit under test, and a response monitor evaluates the test responses.

Different BIST schemes implement different types of TPGs and response monitor circuits. In the most common type of response monitor, test responses are compacted to form signatures. Response signatures are compared with reference signatures, and an error signal indicates a discrepancy. Here we assume this type of response monitor.

Generator types

TPGs can be pseudorandom, exhaustive, or deterministic, depending on the type of test sequence they generate.

Pseudorandom and Exhaustive. Pseudorandom TPGs are by far the most common. However, many circuits contain faults that can be tested by only a few input patterns. It can take a long time for these patterns to appear in a pseudorandom sequence. Consequently, test-sequence lengths and test-application times can be excessive.

The same hardware used for pseudorandom TPGs can be used to apply fully exhaustive tests. Except for very small circuits under test, exhaustive TPGs are not practical.

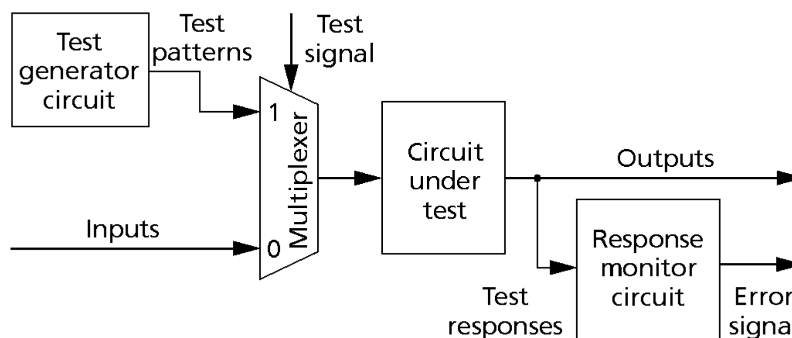

Figure 5. Generic BIST scheme.

Copyright © 1996 IEEE, Inc. All rights reserved.

Deterministic. To address these problems, the test sequence can be made more deterministic. For example, a special seed test can be added to the beginning of a pseudorandom sequence.

However, introducing deterministic tests makes the TPG larger and more complex. A possible solution is to store the tests in a compressed format so they can be produced by a smaller sequence generator. The resulting test patterns must then be decoded by decompression logic. The design goal is to determine a compression method that requires less logic to decompress than the logic it saves in the sequence generator.

At the other end of the BIST system is the response monitor. The goal in compaction is to preserve fault information while reducing the storage requirements of the signatures. Test responses can be compacted in both space and time. In space compaction, each response of the circuit is compacted to reduce the number of bits it contains. In time compaction, a sequence of responses is compacted to produce a single signature value. The most common form of compaction is time compaction via a method similar to the cyclic redundancy check used in serial communication networks like the Internet.

The advantage of time compaction is that only a single signature representing the entire test must be stored. The disadvantage is that errors can only be detected after the entire test has been applied, which limits the applicability of time compaction in on-line testing.

There is another drawback to response compaction: Most compactors do not preserve all fault information. When a response sequence containing errors produces the same signature as a response sequence with no errors, the compactor is said to exhibit aliasing, which clearly reduces the fault coverage of BIST. A number of techniques have been developed to eliminate or reduce the effects of aliasing,[8] all of which add to the area of the circuit.

Core Testing

A core is a large, predesigned standard cell. Popular core types include CPUs, digital signal processors, I/O bus interface modules, and RAMs.

Although standard cells have been used in chip design since the mid-1970s,[9] cores are much larger than these traditional standard cells. Today, cores are typically supplied to designers as a program-like circuit description or model written in a standard hardware description language such as Verilog or VHDL. Each core can have an associated precomputed test set compiled by the core's designer, which may or may not be suitable for testing the core when it is embedded in a larger design.

A core-based ASIC (application-specific IC) design is typically composed of a number of large core modules connected by various interface or "glue" circuits. An example of a very common core type is a microcontroller—a combination of a popular CPU with RAMs, ROMs, and interface modules. A microcontroller is a programmable computer-like control unit.

Figure 6 shows an example of such a core, the VY86C710A from VLSI Technology, which is built around the ARM7 RISC microprocessor. Note the inclusion of logic in the core to support testing.

This microcontroller illustrates another feature: Each major module in a core may itself be another core. By using designs of this kind, a core vendor can offer dozens of versions of microcontrollers based on the same basic CPU design. Some companies even sell customizable microcontroller cores—the customer can choose the core modules to be added to the CPU.

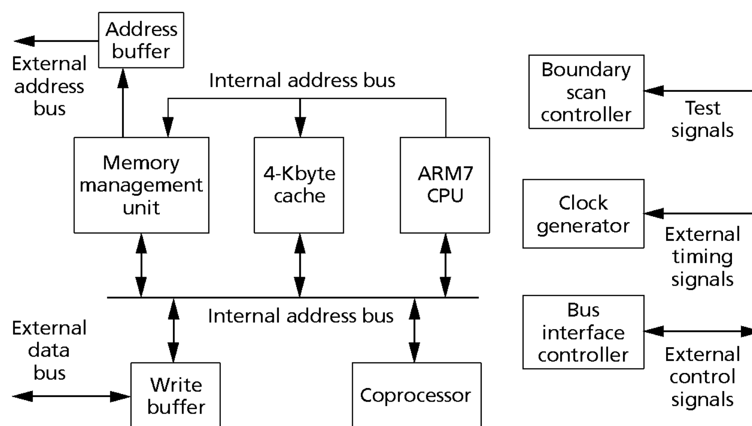

Figure 6. Sample microcontroller core.

Originally published in *Computer*, Nov. 1996, pp. 33–38.

Copyright © 1996 IEEE, Inc. All rights reserved.

It is usually the connections between the cores that make it frustratingly difficult to test core-based designs. Precomputed tests may not exactly match the way a core is used in a particular circuit. For instance, the VY86C710A's memory management unit cannot be accessed directly from the pins of the chip that contains it.

To test a core-based design, the tester must be able to access the inputs and outputs of the cores and weave together the tests of many, possibly heterogeneous, cores into a complete test procedure.

Core types

Cores are known as either *soft* or *hard*. *Soft cores* are high-level or behavioral descriptions that lack full implementation details. They are suitable for both simulation and automatic synthesis. Vendors of soft cores design the modules in HDLs and attempt to verify their correctness, but typically leave their detailed implementation and manufacture to others. The purchasers of a soft core model have complete control over the module's logic-level details because they are responsible for logic synthesis. As a consequence, standard test generation procedures can be used to derive tests for the circuits that employ such cores. However, the user derives no testing benefit from the fact that the core is based on a standard design. Precomputed tests, if available, usually are not guaranteed to provide any particular level of fault coverage, since coverage is implementation dependent.

Hard cores are fully implemented circuit models. If designed and marketed by traditional suppliers of standard cells, they are simply the largest cells in the vendor's library.

Vendors of hard cores are often also chip manufacturers. They consider the design details of the cores to be proprietary, and by extension they consider the tests to be intellectual property as well. Precomputed tests may exist for the cores, but they are not made available to the end user. The intellectual property issues related to the use and testing of cores are being hotly debated at present. Nevertheless, since both the use of cores in design and the importance of testing them are increasing rapidly, we can expect some resolution in the near future.

On the other hand, the technical problems of achieving an acceptable degree of controllability and observability in core-based design may require much more research and standardization effort.

Testing methods

There are two basic methods for testing core-based designs:

1. Propagate test stimulus to the cores through the surrounding circuits and test responses from the core outputs to observation points. For example, PathPlan,[10] Artest,[11] and other tools process precomputed test data in high-level or symbolic form. Such tools can generate tests very efficiently for circuits with certain desirable features. For example, when a regular (data) bus structure is present, all bits of an output bus from a core can be connected to the corresponding inputs of another core or to a primary output bus, thereby facilitating the transmission of test data. Such transmission is also greatly helped by cores and other components that are *transparent* in the sense that test data can be propagated through them without information loss. Circuits with irregular interconnections and nontransparent components are more difficult to test, and frequently must be modified via design-for-testability techniques.

2. Provide direct access to the cores via design-for-testability. This method focuses on providing access to embedded modules by special hardware, which makes test generation much easier[12] Three primary methods for providing direct access to embedded cores have been identified: scan, multiplexing, and BIST. The drawbacks of these three methods are by now familiar: Chip area is increased and performance can suffer. The circuits added for core access are typically not reconfigured from circuits already present, so the penalties are even higher. Moreover, it is difficult to analyze the coverage of BIST, especially the pseudorandom kind, when the cores' logic models are not available for fault simulation.

Conclusion

Testing is a major contributor to the cost of manufacturing and maintaining digital ICs. Well-developed fault models and test generation methods for such circuits are known, and are widely supported by design tools. However, their applicability to today's increasingly fast and complex circuits is limited by practical cost considerations.

Design-for-test techniques, especially scan design and built-in self-test, can provide a satisfactory solution in many instances. Developments in IC technology continue to pose new testing challenges, however; a noteworthy example is the problem of efficiently testing embedded cores.

Copyright © 1996 IEEE, Inc. All rights reserved.

References

1. K.M. Thompson, "Intel and the Myths of Test," *Proc. Int'l Test Conf.,* IEEE CS Press, Los Alamitos, Calif., 1995, keynote address).
2. M. Abramovici, M.A. Breuer and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, 1990.
3. M.H. Schulz, E. Trischler, and T.M. Sarfert, "Socrates: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. Computer-Aided Design*, Jan. 1988, pp. 126-137.
4. I. Pomeranz, L.N. Reddy, and S.M. Reddy, "CompacTest: A Method to Generate Compact Test Sets for Combinational Circuits," *Proc. Int'l Test Conf.,* IEEE CS Press, Los Alamitos, Calif., 1991, pp. 194-203.
5. D. Bhattacharya, B.T. Murray, and J.P. Hayes, "High-Level Test Generation for VLSI," *Computer*, Apr. 1989, pp. 16-24.
6. F.J. Ferguson and J.P. Shen, "A CMOS Fault Extractor for Inductive Fault Analysis," *IEEE Trans. Computer-Aided Design*, Nov. 1988, pp. 1,181-1,194.
7. R.T. Maniwa, "Design for Test Tools," *Integrated System Design*, Sept. 1996, pp. 60-74.
8. K. Chakrabarty, B.T. Murray, and J.P. Hayes, "Optimal Space Compaction of Test Responses," *Proc. Int'l Test Conf.*, 1995, pp. 834-843.
9. G. Persky, D.N. Deutsch, and D.G. Schweikert, "LTX—A Minicomputer-Based System for Automated LSI Layout," *J. Design Automation and Fault-Tolerant Computing,* May 1977, pp. 217-255.
10. T. Murray and J.P. Hayes, "Hierarchical Test Generation Using Precomputed Tests for Modules," *IEEE Trans. Computer-Aided Design*, Sept. 1990, pp. 594-603.
11. J. Lee and J.H. Patel, "Architectural Level Test Generation for Microprocessors," *IEEE Trans. Computer-Aided Design*, Oct. 1994, pp. 1,288-1,300.
12. F. Beenker et al., "A Testability Strategy for Silicon Compilers," *Proc. Int'l Test Conf.,* IEEE CS Press, Los Alamitos, Calif., 1989, pp. 660-668.

Brian T. Murray is a staff research engineer at General Motors R&D Center, where he is investigating cost-effective methods for ensuring the integrity of mission-critical embedded controllers, and a visiting lecturer at the University of Michigan. He is interested in all aspects of embedded system design, especially hardware-software codesign, and verification, on-line testing, core testing, and BIST. Murray received a BA in physics and mathematics from Albion College, an MS in electrical engineering from Duke University, and a PhD in computer science and engineering from the University of Michigan. He is a member of IEEE, IEEE Computer Society, ACM, and Sigma Xi.

John P. Hayes is a professor in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor. He has also served on the faculty of the University of Southern California. He teaches and conducts research in the areas of computer-aided design verification and testing, computer architecture, VLSI design, and fault-tolerant computing, and was the founding director of Michigan's Advanced Computer Architecture Laboratory. He is the author of several books, including Computer Architecture and Organization (McGraw-Hill, 2nd ed., 1988), Layout Minimization for CMOS Cells (Kluwer, 1992), and Introduction to Digital Logic Design (Addison-Wesley, 1993), as well as more than 150 technical papers. Hayes received a BE from the National University of Ireland, Dublin, and an MS and a PhD from the University of Illinois, Urbana-Champaign, all in electrical engineering. He is a fellow of IEEE, and a member of ACM and Sigma Xi.

Contact Murray or Hayes at the University of Michigan, Electrical Engineering and Computer Science Dept., 1301 Beal Ave., Ann Arbor, MI 48109-2122; bmurray@gmr.com; jhayes@eecs.umich.edu.

Originally published in *Computer*, Nov. 1996, pp. 33–38.

Copyright © 1996 IEEE, Inc. All rights reserved.