LOGIC TESTING

The rapid and copious advances in semiconductor technology have enabled integrated circuit (IC) densities (number of components and logic gates per chip) to grow phenomenally. This has allowed the designers to implement a multitude of complex logic functions in digital hardware, often on a single chip. It is in the vital interest of both the producer and the end user to ensure that such a complex digital system functions correctly for the targeted application. It is also of interest to evaluate the reliability of a product, that is, to know whether the product will continue to function correctly for a long time. To guarantee functional correctness and reliability of a product, the producers and end users rely on *testing*.

In its broadest sense, testing means to examine a product, to ensure that it functions correctly and exhibits the properties it was designed for. Correct functioning of an electronic computer system relies on fault-free hardware and software components. The subject of this article is testing digital logic circuits using test equipment and related testing aides, so as to detect malfunction and incorrect behavior.

DIGITAL LOGIC CIRCUITS

Digital logic circuits can be classified into *combinational* and *sequential* circuits. Combinational logic circuits are digital circuits characterized by the fact that the logic values computed at their outputs are a function only of the present input values. Sequential circuits are digital systems whose outputs depend on the present inputs and also on the previous input values. Although combinational logic circuits consist of a network of interconnected logic gates, sequential circuits also contain memory elements which remember the history of the previous input patterns. These memory elements are implemented as *registers* or *flip-flops,* and their unique configurations represent the *states* of the sequential circuit. Thus, the outputs of a sequential circuit depend on the present inputs and the present internal state stored in these memory elements. Because of their inherent sequential nature, sequential circuits are harder to test than combinational circuits because more information is required to identify their faulty operation.

CONVENTIONAL TEST METHODS AND TEST ENVIRONMENTS

Figure 1 shows a conceptual environment for testing a logic circuit. The unit under test (UUT) is connected to its tester via an interface circuitry which consists of drivers, receivers, contact probes, and cable connections. In its most basic form, testing consists of applying *stimuli* to a UUT and comparing its *responses* with the known fault-free behavior. To obtain fault-free responses, test engineers often stimulate a verified fault-free unit simultaneously with the UUT using the same test patterns. Instead of an actual circuit, a hardware emulation or a software model of the designed system can also be used to obtain fault-free responses. Fault-free responses may also be available as the functional specifications of the product.

With increasing circuit densities, large and complex digital circuits are being assembled on a chip. This has led to greater difficulties in accessing individual circuit components. To cope

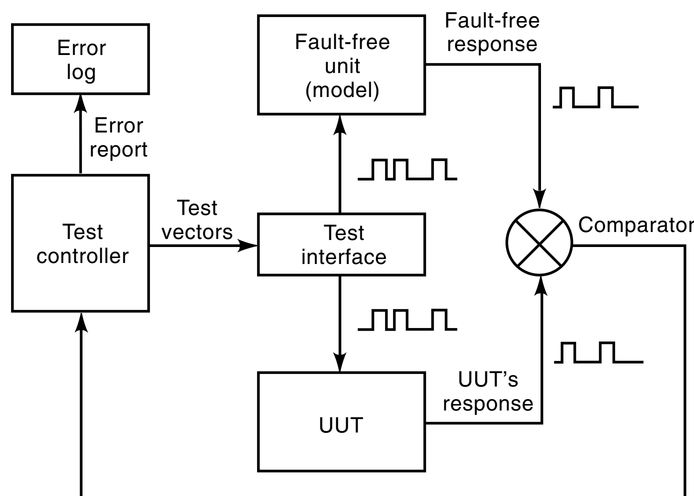


Figure 1. A typical testing environment applying test patterns to a UUT via a test interface and comparing its output responses with the fault-free responses.

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*. Copyright © 1999 John Wiley & Sons, Inc.

with this problem, there have been continuing efforts to develop test points within the circuits and to develop miniature probes to access circuit components via these test points. Because logic circuits perform many functions, process a large amount of data, and are increasingly complex, it has also become impossible for test engineers to test them manually. Such problems, in combination with the advancement of computer technology and data acquisition systems, have led to the emergence of *automatic test equipment* (ATE). ATE uses test programs to automatically compute a series of stimulus patterns, called *test vectors,* and applies these vectors to the inputs of the UUT through the test interface. ATE acquires the *responses* from the outputs of the UUT and automatically compares these responses with the responses expected of an ideal (fault-free) unit. If these responses are not in agreement, errors are registered automatically.

FAULT MODELS AND TESTING TYPES

An instance of an incorrectly operating UUT is called an *error* or a *fault*. Incorrect and erroneous operation can be attributed to *design errors, fabrication errors,* or other *physical defects.* Examples of design errors are inconsistent specifications, logical errors or bugs in the design, and violations of design rules. Fabrication errors include faulty and incorrect components, incorrect wiring, and "shorts" or "opens" caused by improper soldering. Physical defects generally occur because of component wear out during the lifetime of a system. For instance, aluminum wires inside an integrated circuit (IC) thin out with time and may eventually break because of a phenomenon called *electromigration*. Environmental factors, such as humidity, heat, and vibrations, accelerate component wear and tear.

In general, direct mathematical treatment of physical failures and fabrication defects is not feasible. Thus, test engineers model these faults by *logical faults,* which are a convenient representation of the effect of physical faults on system operation.

Such fault models assume that the components of a circuit are fault-free and only their interconnections are defective. These logical faults can represent many different physical faults, such as opens, shorts with power or ground, and internal faults in the components driving signals that keep them stuck-at a logic value. A *short* results from unintended interconnection of points, while an *open* results from a break in a connection. A short between ground or power and a signal line can result in a signal being *stuck* at a fixed value. A signal line when shorted with ground (power) results in its being *stuck-at-0* (*stuck-at-1*) and the corresponding fault is called a *s-a-0* (*s-a-1*) fault.

Figure 2 illustrates the effect of an *s-a-0* fault at line A on the operation of a circuit. An input signal to an AND gate when shorted to ground (*s-a-0*) results in its output always being *s-a-0.* This, if line A is *s-a-0,* irrespective of the values at all other inputs of the circuit, output Z will always evaluate incorrectly to logic value 0. In such a way, the presence of a stuck-at fault may transform the original circuit to one within a different functionality. Testing for physical defects and failures is carried out by applying input vectors that excite the stuck-at faults in the circuit and propagate their effect to the circuit outputs. The observed responses to the test vectors are compared with the expected fault-free responses to identify fault operation and diagnose physical defects.

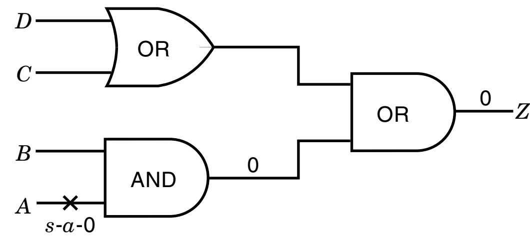


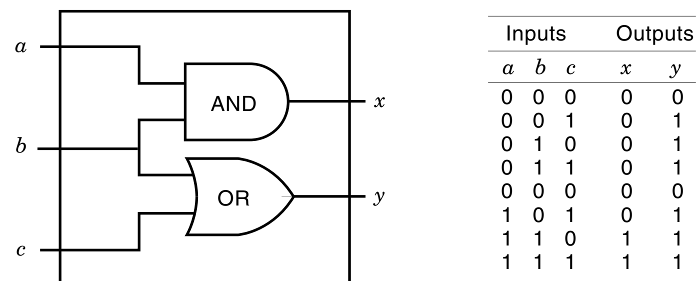
Figure 2. Faulty operation of a circuit due to a stuck-at-0 fault at line A.

Testing for fabrication defects and errors, generally known as *burn-in* or *acceptance testing,* is usually performed by the manufacturer. Testing for physical defects and failures is beyond the scope of this article. The interested reader is referred to Ref. 1 for a thorough treatment of the subject.

Design verification testing is carried out to test for design errors. It can be performed by a testing experiment on an appropriate model of the designed system. These models are usually software representations of the system in terms of data structures and programs. Examples of such models are binary decision diagrams (BDDs), finite-state machines (FSMs), and iterative logic arrays (ILAs). Such a model is exercised by stimulating it with input signals. The process is called *logic simulation.* Usually such models are functional models, that is, they reflect the functional specifications of the system and are independent of the actual implementation. Hence, the process of testing a digital logic circuit with respect to its functional specification is called *functional testing*.

Functional Testing

There is no established definition of functional testing per se. In its most general sense, functional testing means testing to ascertain whether or not a UUT performs its intended functions correctly (2). Thus, functional testing validates the correct operation of a system with respect to its functional specification. Functional testing is targeted toward a specific fault model or is performed without any fault models. In the former approach, tests are generated for a UUT that detect faults defined by such models. The latter tries to derive tests based on the specified fault-free behavior. Another approach defines an implicit fault model (also known as the *universal fault model*) which assumes that *any* fault can occur. Functional tests detecting any fault are said to be *exhaustive* because



Inputs			Outputs	
a	*b*	*c*	*x*	*y*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
0	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Figure 3. A combinational circuit (a) and its truth table (b). All possible input combinations are required for exhaustive testing.

Table 1. Required Vectors for Pseudoexhaustive Testing Are a Subset of the Set of Vectors Required for Exhaustive Testing

Inputs			Outputs	
a	*b*	*c*	*x*	*y*
0	0	0	0	0
0	1	0	0	1
1	0	1	0	1
1	1	1	1	1

they completely exercise fault-free behavior. However, because of their exhaustive nature, such tests are impractical for large circuits. It is often possible to use some knowledge about the structure (or functionality) of the circuit to narrow the universe of detected faults. Test sets thus obtained are significantly smaller and are *pseudoexhaustive*.

EXHAUSTIVE AND PSEUDOEXHAUSTIVE TESTING OF COMBINATIONAL LOGIC CIRCUITS

Exhaustive tests detect all possible faults defined by the universal fault model. In a combinational circuit with n inputs, there are 2^n possible input signal combinations. Hence, to test a combinational circuit exhaustively, all 2^n possible input vectors need to be applied to the circuit. The exponential growth of the required number of vectors in the number of inputs makes exhaustive testing impractical. However, pseudoexhaustive testing methods significantly reduce the size of the test set and detect a large subset of all possible faults.

As an example, consider a circuit with three inputs and two outputs, shown in Fig. 3(a). To test this circuit exhaustively as a "black box" without any knowledge of its structure, all of the vectors shown in Fig. 3(b) have to be applied. On the other hand, if some information about the underlying structure of the circuit and the input/output dependence is available, only a subset of the vectors may be sufficient to test the circuit pseudoexhaustively. For the example circuit shown in Fig. 2(a), the output x depends only on inputs a and b and does not depend on input c. Similarly, output y depends only on inputs b and c. Because of such a *partial dependence* of outputs on the inputs, it is sufficient to test output x exhaustively with respect to inputs a and b, and similarly output y with respect to inputs b and c. Thus, as shown in Table 1, just four vectors are required to test this circuit pseudoexhaustively. However, a fault caused by a "short" between input lines a and c (known as a *bridging fault*) cannot be detected by the test set shown in Table 1. Except for such faults, all faults defined by the universal fault model can be detected.

The previous method, however, cannot be applied to *total-dependence* circuits, where at least one primary output depends on all primary inputs. In such cases, circuit *partitioning techniques* can be used to achieve pseudoexhaustive testing. Using partitioning techniques, circuits are partitioned into *segments* so that the outputs of the segments depend only on their local inputs. Then each segment is exhaustively tested with respect to its inputs. Figure 4 shows a circuit partitioned into segments. Each segment can be exhaustively tested with respect to their local inputs. In Ref. 3 extensions of partitioning techniques were applied for pseudoexhaustive testing of a commercial 4-bit arithmetic and logic unit (ALU) with 14 inputs. Pseudoexhaustive testing required just 356 test vectors, a small fraction of the 2^{14} vectors required for exhaustive testing.

Functional testing is used by manufacturers and also by field-test engineers and end users of systems. Manufacturers do not normally supply structural models or implementation details of a product. Usually, only the functional specifications of a product are provided to the users. Thus end users rely on functional testing methods (or variants) to verify whether a product *conforms* to its particular set of specifications.

SEQUENTIAL CIRCUIT TESTING

Testing of sequential circuits is a much more involved process compared with testing of combinational circuits because the response of a sequential circuit is a function of its primary inputs and also of its internal states. In general, it is custom-

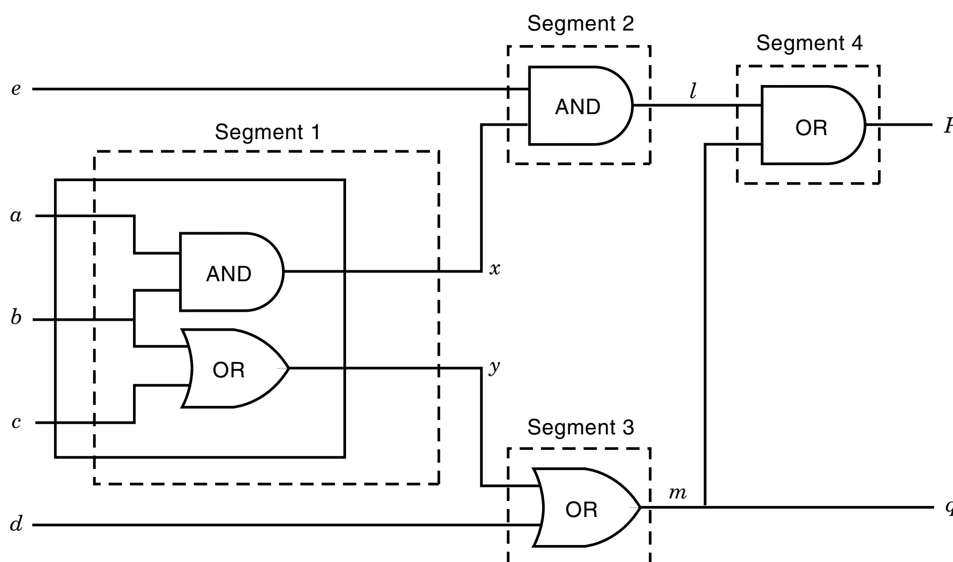
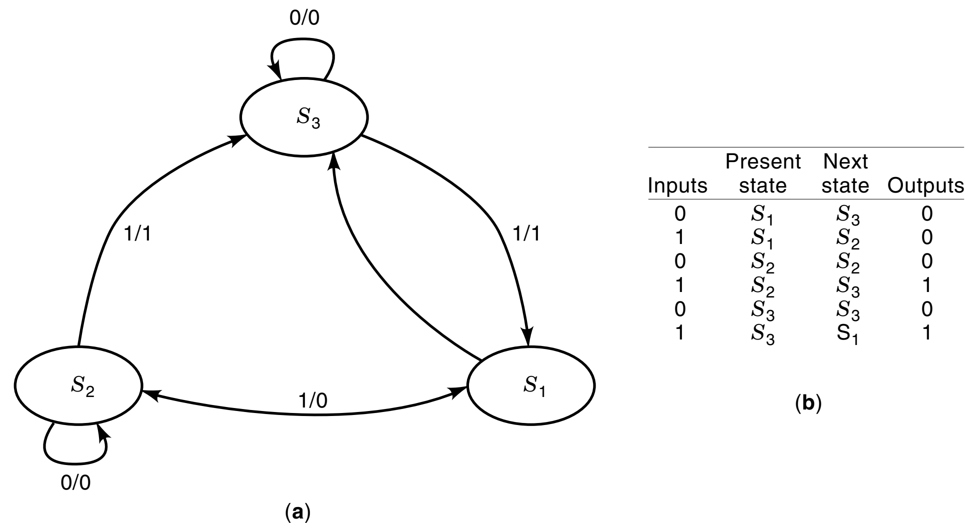


Figure 4. Circuit partitioning into segments for pseudoexhaustive testing.



Inputs	Present state	Next state	Outputs
0	S_1	S_3	0
1	S_1	S_2	0
0	S_2	S_2	0
1	S_2	S_3	1
0	S_3	S_3	0
1	S_3	S_1	1

Figure 5. FSM representation: (a) state-transition graph; (b) state-transition table.

ary to model a sequential circuit as a finite automaton or a *finite-state machine* (FSM). An FSM can be represented by a state-transition table (STT), or by its equivalent state-transition graph (STG). Use of such representations allows the designers and test engineers to better understand the behavioral characteristics and functionalities of sequential circuits. It also allows them the flexibility to apply various Boolean and mathematical transformations without any explicit knowledge of the underlying technology. Before delving into the details of sequential circuit testing, it is important to understand fundamental concepts about finite-state machine operation.

FSM Representation

Figure 5 shows a graphical and a tabular representation of a finite-state machine. The vertices in the STG represent the states of the machine and the arcs represent the transitions between the states. In response to a set of inputs, a finite-state machine transits from its *current* internal state (also called present state) to a *next* state and produces a set of outputs. The states of an FSM are assigned binary encodings and are physically implemented with synchronous delay elements, called flip-flops or registers. Each state of the machine is represented by the set of values in the registers. In such a representation, there in an inherent assumption of *synchronization* that is not explicitly represented in the STG or the STT. Because of this synchronization, the data stored in the registers is sampled by a signal called *clock,* the next state is entered and the output is produced.

A canonical structure of a synchronous sequential circuit is shown in Fig. 6. It is composed of a combinational logic component whose present state inputs (y) and the next state outputs (Y) are connected by a feedback loop involving the state registers. The primary inputs are represented as x and the primary outputs as z. In response to a known input sequence, the succession of states traversed by an FSM and the output responses produced by the machine are specified *uniquely* by its state representation (STT or STG). Thus, under the universal fault model, faults or errors in sequential circuits are accounted for by any fault that modifies the state-transition representation of the underlying FSM. To detect faulty behavior and identify the faults in sequential circuits, test engineers apply various input sequences to compare the observed output values with the known responses derived from the state table. Such experiments are known as *state-identification* and *fault-detection* experiments.

Fault-Detection and State-Identification Experiments

Machine-identification experiments are concerned with the problem of determining whether an n-state machine is distinguishable from all other n-state machines. These experiments are also used to determine whether a machine is operating correctly with respect to its specifications. In such experiments, a sequential circuit is viewed as a "black box," and by applying certain input sequences and observing the output responses, the experimenter has either to identify the states of the machine or detect its faulty behavior.

The experiments designed to identify the states of an FSM distinguish one state of the machine from the other. They are known as *state-identification* or *state-distinguishing* experiments. In such experiments, it is often required to drive the machine either to a uniquely identifiable state, or to a prespecified state. A machine is made to visit different states by applying various input sequences, and these states are determined by observing the output responses of the machine. It is customary to call the state, in which the machine resides before applying any input sequence, the *initial* state. The state in which the machine resides after applying an input se-

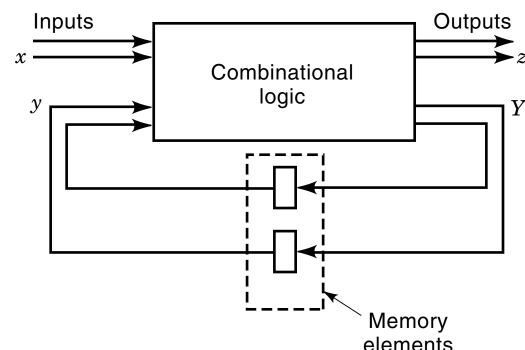


Figure 6. A canonical representation of an FSM.

quence is called the *final* state. Then this final state is used as a "reference point" for further experiments.

Homing experiments are generally conducted to bring a machine from an unknown state to a uniquely identifiable final state. In these experiments, a sequence of inputs is applied to the machine to bring it to a final state. The final state in which the machine resides is identified uniquely from the machine's response to the input sequence. Such an input sequence is known as a *homing sequence*. Consider the example machine M and its state table shown in Table 2(a). A homing sequence for this machine is $X_h = \langle 101 \rangle$. The final state of the machine is uniquely determined from the response of the machine M to this input sequence. As can be seen from Table 2(b), if the output response is $\langle 000 \rangle$, then it can be said, beyond doubt, that machine M is in final state S_0. Similarly, the output response $\langle 101 \rangle$ means that the machine is in final state S_3. Though a machine may possess more than one homing sequence, the shortest one is usually of interest.

To initialize a machine to a known state, a *synchronizing sequence* X_s is applied. This sequence takes the machine to a prespecified final state, regardless of the output or the initial state. For example, the sequence $X_s = \langle 10101 \rangle$ synchronizes the machine M to state S_3, regardless of its initial state. Not all machines, however, possess such a sequence. The application of a *distinguishing sequence* X_d produces a different output sequence for each initial state of the machine, and thus distinguishes among its different states. Hence, the state of the machine before applying X_d is uniquely identified by its output response to X_d. Note that every distinguishing sequence is also a homing sequence, but the converse is not always true. A comprehensive treatment of state-identification experiments can be found in Refs. 5 and 6.

The input sequences described previously are helpful for identifying and differentiating the states of a machine and also to detect the machine's faulty behavior. Any input sequence that detects any fault defined by the universal fault model must distinguish a given n-state sequential machine from all other machines with the same inputs and outputs and at most n-states (7). The fault-detection experiments, designed to identify faulty behavior of the machines, are also called checking experiments and consist of the following three phases:

- initializing the machine to a known starting state by using a synchronizing sequence;
- verifying that the machine has n states;
- verifying every entry in the state table by exercising all possible transitions of the machine.

For the first part of the experiment, initialization is accomplished by using the synchronizing sequence, which brings the machine to a unique state S. Now this state becomes the initial state for the rest of the experiment. To check whether or not the machine has n states, it is supplied with appropriate input sequences that cause it to visit all possible states. Each state is distinguished from the others by observing the output responses to the distinguishing sequence. During the course of this testing experiment, if the machine has not produced the expected output, it is concluded that a fault exists. Finally, to conclude the experiment, it is required to verify every state transition. The desired transitions are exercised by applying the appropriate input, and each transition to a state is verified with the help of the distinguishing sequence.

Fault-detection experiments for machines that do not have distinguishing sequences are complicated, and the resulting experiments are very long. Thus, the design of "easily testable" sequential circuits that possess some distinguishing sequence has been a subject of extensive research.

The previous methods for verifying the correctness of sequential machines are based on deriving the information from the state table of the circuit. These methods are exhaustive, and thus have practical limitations for large circuits. For sequential circuits that can be structured as iterative logic arrays (ILAs), pseudoexhaustive testing techniques can be used to test them efficiently. Recently, the problem of verifying the correctness of sequential machines has received a lot of attention. Formal methods have been developed to verify the equivalence of sequential circuits against their finite-state machine models. A recent text (8) is a good source of information on the subject.

Table 2. Machine *M*: (a) State Transition Table, (b) Response to its Homing Sequence 101

Inputs	Present State	Next State	Outputs
0	S_0	S_3	0
1	S_0	S_1	0
0	S_1	S_1	0
1	S_1	S_0	0
0	S_2	S_0	0
1	S_2	S_3	1
0	S_3	S_2	0
1	S_3	S_3	1

(**a**)

Initial State	Response to Sequence 101	Final State
S_0	000	S_0
S_1	001	S_3
S_2	101	S_3
S_3	101	S_3

(**b**)

DESIGN FOR TESTABILITY AND SELF-TEST TECHNIQUES

For finite state machines with a large number of states, distinguishing and synchronizing sequences become unreasonably long, resulting in long test application times. Hence, it is desirable to design circuits in such a way that they are easier to test. Small circuit modifications can aid in the testing process by providing easier or direct access to test points, can shorten the length of input test patterns, reduce test application time, while preserving the intended design behavior. Techniques which modify the circuit to make it easily testable are commonly called *design for testability* (DFT) techniques.

Scan Test

One of the most widely used DFT techniques is *scan design.* The rationale behind the scan design approach is to convert a sequential circuit into a combinational one in order to make it easier to test. This is carried out by modifying the registers (flip–flops) to enable their access directly through their inputs

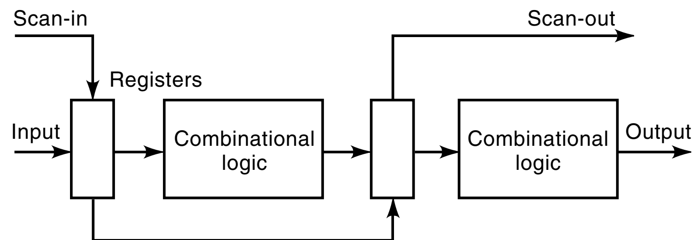


Figure 7. Scan registers connected serially in serial-scan chain.

and outputs. These registers are called scan registers. Circuits with scan registers operate in two modes: (i) the normal mode of operation, and (ii) the test mode. In the test mode, the test equipment has virtually direct access to the registers which enables the application of test vectors directly on the combinational logic. Since the number of input and output (IO) pins on a chip is limited, it is impossible to directly access all the registers through the IO terminals. Thus, scan registers are chained together as a single serial shift register, as shown in Fig. 7. Test vectors are shifted serially into the registers via the *scan-in* input pin, and the output responses to these vectors are shifted out via the *scan-out* pin.

However, it is not always desirable to make all the registers scannable. Scanning all the registers adversely affect the area and performance of the circuit due to the necessary modifications required to accommodate the complete scan chain. The extensive serial shifting of test patterns and responses also results in unacceptable length of the resulting tests. *Partial scan* provides a trade-off between the ease of testing and the costs associated with scan design. In partial scan, only a subset of registers is selected for scan, which limits the increase in area and delay of the circuit. However, the key problem in partial scan design is the selection of scan registers. A lot of research has been devoted to define the criteria to guide the selection of scan registers. References 9–12 are a good source of information on the subject.

Scan testing techniques have also been applied to test printed circuit boards. This technique, called the *boundary scan* technique, has been standardized (13) to ensure compatibility between different vendors. It connects the input and output pins of the components on a board into a serial scan chain. During the normal operation, the boundary scan pads act as normal input-output pins. In the test mode, test vectors are serially scanned in and out of the pads, thus providing direct access to the boundary of the components to be tested.

Built-In Self Test

Built-in self test (BIST) techniques rely on augmenting a circuit so that it allows itself to generate test stimuli and observe output responses in order to validate correct operation. Figure 8 illustrates the general format of a BIST structure. The stimulus generator is responsible for generating test sequences. Exhaustive, random, and pseudorandom approaches are used to generate the test stimuli. In the exhaustive approach, all possible input vectors are generated automatically. An N-bit counter is an example of an exhaustive test pattern generator. Random test stimulus generator applies randomly chosen subset of possible input patterns. A *pseudorandom sequence generator* (PRSG) implements a polynomial of some length N. It is constructed from a set of registers connected in a serial fashion, called the *linear feedback shift register* (LFSR). Outputs of certain shift bits are XORed and fed back to the input of the LFSR. An N-bit LFSR cycles through $2^N - 1$ states before repeating the sequence, producing a seemingly random sequence.

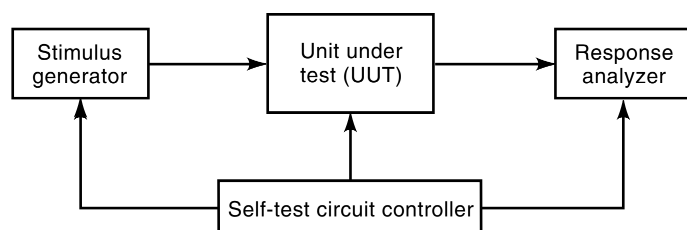


Figure 8. A typical built-in self-test structure.

The response of the analyzer can be implemented as a comparison between the generated response and the expected response, and stored in an on-chip memory. However, this requires excessive memory and thus results in large area overheads. Usually, the responses are *compressed* and then stored into memory. The compressed response is also known as a *signature* and hence the approach is called *signature analysis.* A fault in the logic circuit causes its signature to change from a known good value which indicates the faulty operation.

Self-testing techniques are widely used in testing regular structures such as memories. Memory tests include the reading and writing of a number of different patterns into and from the memory using alternating addressing sequences. With a minimal area overhead, this test approach is built into the integrated circuit itself which significantly improves the testing time and minimizes external control.

CONFORMANCE AND INTEROPERABILITY TESTING

Building a system involving products from a number of different vendors is a challenging task, even when the components are supposed to conform to the appropriate systems standards. Nowadays, digital systems are so notoriously complex that even the functional specifications provided by manufacturers are not sufficient to determine the *interoperability* of the equipment. This problem has strongly affected the technology industries that provide multivendor products, like personal computers, computer peripherals, and networking solutions. With the emergence of the “information age,” the need for interconnection and interoperability of information technology (IT) products, such as data communication and networking hardware, implementations of communication protocols and other related software products, has also grown manifold. *Conformance testing* combined with *interoperability testing* greatly reduces the problems associated with building multivendor systems.

The term *conformance* refers to meeting the specified requirements. In conformance testing, a product is tested using specified test cases to verify whether or not it violates any of the specified requirements and to validate that it behaves consistently with respect to the options (or functions) that it is said to support. In conformance testing, a product is tested for each specification that it supports. Test engineers often use ATE to automate the processes of test purpose and test

case generation and also to validate, compile, and maintain the test suites. The result of conformance testing is a test report which specifies whether or not the given product passes each of the test cases. Conformance testing is carried out by vendors, procurers, or independent testing laboratories.

Interoperability testing provides evidence whether a specific product can be made to "interface" effectively with another product implementing the same specifications. Vendors normally perform such tests to check interoperability before a product is released. Interoperability testing is also used by major procurers to check the acceptability of equipment combinations that they wish to buy.

Acknowledging the previously mentioned problems of conformance and interoperability, Open Systems Interconnection (OSI) standards have been developed to achieve interoperability between equipment from different manufacturers and suppliers. International Standard (IS) 9646 is a standard devoted to the subject of conformance testing implementations of OSI standards. IS 9646 prescribes how the base standards have to be written, how to produce test suites for these standards, and how the conformance testing process has to be carried out. A comprehensive description of IS 9646 can be found in Ref. 4 with particular applications to conformance testing of communication protocols.

PERSPECTIVES

Testing of logic circuits has been an actively researched area for more than three decades. A high degree of automation has been achieved, new theories and techniques have been proposed, and many algorithms and tools have been developed to facilitate the testing process. However, with the unprecedented advances in device technologies and growth in circuit size, testing is becoming increasingly difficult. The high cost and limited performance of test equipment and the high cost of test generation are other problems affecting test engineers. For such reasons, *design for testability* and *self-checking designs* are becoming more and more attractive to the testing community.

BIBLIOGRAPHY

1. M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design,* Piscataway, NJ: IEEE Press, 1990.
2. F. F. Tsui, *LSI/VLSI Testability Design,* New York: McGraw-Hill, 1987.
3. E. J. McCluskey and S. Bozorgui-Nesbat, Design for autonomous test, *IEEE Trans. Comput.,* **C-33**: 541–546, 1984.
4. K. G. Knightson, *OSI Protocol Conformance Testing: IS 9646 Explained,* New York: McGraw-Hill, 1993.
5. Z. Kohavi, *Switching and Finite Automata Theory,* New York: McGraw-Hill, 1970.
6. A. Gill, State identification experiments in finite automata, *Inf. Control,* **4**: 132– 154, 1961.
7. A. D. Friedman and P. R. Menon, *Fault Detection in Digital Circuits,* Englewood Cliffs, NJ: Prentice-Hall, 1971.
8. G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms,* Norwell, MA: Kluwer, 1996.
9. V. D. Agarwal et al., A Complete Solution to the Partial Scan Problem, *Proc. Int. Test Conf.,* 1987, pp. 44–51.
10. K. T. Cheng and V. D. Agarwal, An Economical Scan Design for Sequential Logic Test Generation, *Proc. Int. Symp. Fault-Tolerant Comput.,* 1989, pp. 28–35.
11. V. Chickermane and J. H. Patel, An Optimization Based Approach to the Partial Scan Design Problem, *Proc. Int. Test Conf.,* 1990, pp. 377–386.
12. P. Kalla and M. J. Ciesielski, A Comprehensive Approach to the Partial Scan Problem using Implicit State Enumeration, *Proc. Int. Test Conf.,* 1998.
13. IEEE Standard 1149.1, IEEE Standard Test Access Port and Boundary-Scan Architecture, *IEEE Standards Board,* New York.

PRIYANK KALLA
MACIEJ J. CIESIELSKI
University of Massachusetts at Amherst

LOGIC TESTING. See AUTOMATIC TESTING.

LOG-PERIODIC ANTENNAS. See DIPOLE ANTENNAS.

LOG-STRUCTURED FILE SYSTEMS. See BATCH PROCESSING (COMPUTERS).