From: EE Handbook, CRC Press, 2005

Digital IC Testing: An Introduction

Micaela Serra
Dept. of Computer Science
University of Victoria
Victoria, B.C. Canada
http://www.cs.uvic.ca/~mserra
From: EE Handbook, CRC Press

In this chapter we give an overview of digital testing techniques with appropriate reference to material containing all details of the methodologies and algorithms. First, we present a general introduction of terminology, a taxonomy of testing methods and of fault models. Then we discuss the main approaches for the generation of test patterns, both algorithmically and pseudo-randomly, concluding with an introduction to signature analysis and Built-In Self-Test.

Taxonomy and Definitions

Technology advances are causing the density of circuits to increase continually, while the number of I/O pins remains small in comparison. This causes a serious escalation in complexity, and testing is becoming a major cost to industry (estimated as up to 40% of the total production cost). Moreover, digital devices are now ubiquitously present in many portable electronics used for safety critical applications, such as biomedical applications. Ensuring their reliability through proper testing is crucial. Devices should be tested before and after packaging, after mounting on a board, and periodically during operation. Different methods may be necessary for each case, but it is of the utmost importance that defective units should be detected as soon as possible in the production chain. The *rule-of-ten* is often invoked: a detection cost *C* at the component level becomes a cost of *10 C* at the board level, a cost of *100 C* for a system, and up to *1,000 C* in the field. In general, by *testing* we imply the means by which some qualities or attributes are determined to be fault-free or faulty. The main purpose of testing is to detect malfunctions (Go/NoGo test), and only subsequently, to increase yield and/or change a production process. One may also be interested in fault diagnosis to narrow down the actual location of the malfunction.

The evaluation of the reliability and quality of a digital device is commonly called *testing*, yet it comprises distinct phases that are usually kept separate, both in the research community and in industrial practice.

1. *Verification* is the initial phase at design time when one ensures the match of requirements specifications to the current functionality, that is, a digital device needs to be verified for correctness. Verification checks that all design rules are adhered to. More generally, this type of functional testing checks that the circuit: (a) implements what it is supposed to do and (b) does not do what it is not supposed to do. Both conditions are necessary. This type of evaluation uses a variety of techniques, including logic verification with the use of hardware description languages, full functional simulation, and generation of functional test vectors. We do not discuss verification techniques here.

2. *Testing* more precisely refers to the phase when one must ensure that only defect-free production devices are shipped, and faults arising from manufacturing are detected. Testing methods must: (a) be fast enough to be applied to large numbers of devices during production; (b) take into consideration possible access to large expensive external tester machines; and (c) consider whether the implementation of Built-In Self-Test (BIST) proves to be advantageous. In BIST, the circuit is designed to include its own self-testing extra circuitry and thus can signal directly, during testing and in the field, its possible failure status. Of course, this involves a certain amount of area overhead and trade-offs must be considered. The development of appropriate testing algorithms and their tool support can require a large amount of engineering effort, but one must note that it may need to be done only once per design. The speed of application of the method to many copies of the devices can be of more importance.

If many defects are present during the manufacturing process, the final yield is lowered. Estimates can be derived for the relationship between manufacturing yield, effectiveness of testing (fault coverage) and defect level remaining after test [Williams, 1986]. Let Y denote the yield, which is some value between 1 (100% defect-free production) and 0 (all circuits faulty after testing), and is approximated by the ratio of "good devices" over "total devices". Let FC be the fault coverage, calculated as the percentage of detected faults over the total number of detectable modeled faults (see below for fault models). The value of FC ranges from 1 (all possible faults detected) to 0 (no useful testing done). We are interested in estimating the final defect level (DL) after test, defined as the probability of shipping a defective product. That is, DL measures the number of bad devices which pass all tests; its value is expressed as number of defects per million (DPM). It has been shown that tests with high fault coverage for certain fault models also have high defect coverage. The empirical equation is: $DL = \mathbf{1} - Y^{(\mathbf{1}-FC)}$. Plotting this equation gives interesting and practical results. For example, if a value of DPM = 300 (that is, DL = 0.0003 = 0.03%) and a value of Y = 0.5 are desired, then it must be that $FC = 1 - (\log(1 - DL) / \log Y) = 0.999567$, which is 99.957%. Conversely, with a similar desired yield Y = 0.5 and FC = 0.9, then $DL = 1 - 0.5^{(1-0.9)} = 0.06697$ which implies that about 6.7% of shipped devices are defective. The main conclusion to be drawn is that a very high fault coverage must be achieved to obtain any acceptable defect level value, and manufacturing yield must be continually improved to maintain the reliability of shipped products.

Most testing techniques focus on combinational circuits only. This is a realistic assumption based on designing sequential circuits by partitioning the memory elements from the control functionality in the combinational modules. This general approach is one of the methods in design for testability (DFT) (see the next Chapter). DFT can encompass any design strategy aimed at enhancing the testability of a circuit. Scan design is the best-known implementation for separating the latches from the combinational modules, such that some of the latches can also be reconfigured and used as either tester units or as input generator units (essential for built-in testing).

Figure 1 shows the taxonomy for testing methods. The main division is between **on-line** and **off-line**. In the former, each input/output word from a circuit can be tested during normal operation, usually implying that the circuit has been augmented to contain some embedded coding scheme to provide the error detection. In the latter, the circuit must suspend normal operation and enter a "test mode," at which time the appropriate method of testing is applied. Off-line testing can be executed either through external testing, for example a tester machine external to the circuitry, or through the use of BIST.

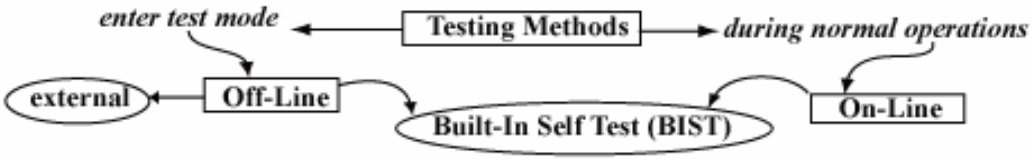

Figure 1: Taxonomy of Testing Methods

Fault Models

At the defect level, an enormous number of different failures could be present and it is totally infeasible to analyze them as such. Thus failures are grouped together with regards to their *logical* fault effect on the functionality of the circuit, and this leads to the construction of logical fault models as the basis for testing algorithms and the evaluation of fault coverage [Abramovici et al., 1992; Jha et al., 2003]. More precisely, a *fault* denotes the physical failure mechanism, the *fault effect* denotes the logical effect of a fault on a signal-carrying net, and an *error* is defined as the condition (or state) of a system containing a fault (deviation from correct state). Faults can be further divided into classes according to their characteristics, as shown in Figure 2. Here we discuss only **permanent** faults, that is, faults in existence long enough to be observed at test time, as opposed to *temporary* faults (transient or intermittent), which appear and disappear in short intervals of time, or *delay* faults, which affect the operating speed of the circuit.

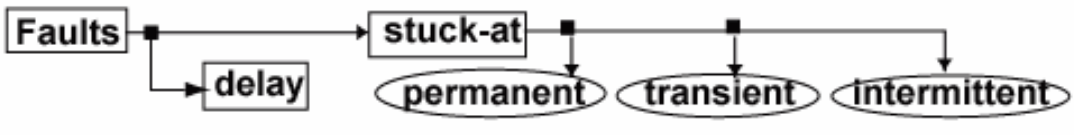

Figure 2: Fault Characteristics

The fundamental fault model is a **stuck-at fault**, which implies the fault effect to be a line segment stuck at logic 0 or 1 (*stuck-at 0* or *stuck-at 1*). Testing may consider single or multiple stuck-at faults and Figure 3 shows an example for a simple circuit. The fault-free function is shown as Z, while the faulty output functions, under the occurrence of the

single stuck-at faults of either line *a* stuck-at 0 (a/0) or of line *b* stuck-at 1 (b/1), are shown as Z*. A stuck-at fault needs a single appropriate input pattern to stimulate the fault by controlling the inputs and making the faulty output observable, and the goal of test patterns generation algorithms is to find such settings for all detectable faults.

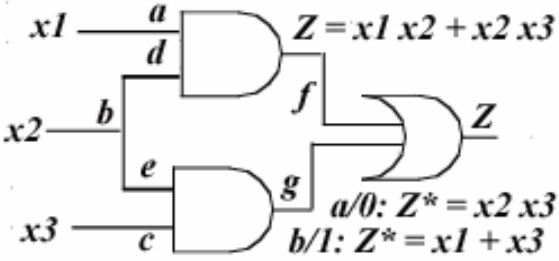

Figure 3: Stuck-at Faults

On the other hand, a delay fault defines the fault effect to be *slow-to-rise* (from 0 to 1) or *slow-to-fall* (from 1 to 0), such that the final value may indeed be correct, but outside the timing parameters expected. To detect such faults, one must apply two patterns as stimuli: the first to set a line at a certain value and the second to change that value. This, of course, increases the complexity of a fault detection algorithm.

Even considering only single stuck-at faults, not all possible faults need to be explicitly tested, as many have the same fault effect and are indistinguishable. For example, an input stuck-at-0 fault for an AND gate has the same logical effect as the output stuck-at-0, and one needs only test for either one. *Fault collapsing* is the process of reducing the total number of faults to be examined using fault equivalence classes based on fault effects. Table 1 shows the main fault equivalence classes for gates.

Table 1: Fault Equivalence

Gate	Fault	Equivalent to:	
AND	any input/0	← →	output/0
OR	any input/1	← →	output/1
NAND	any input/0	← →	output/1
NOR	any input/1	← →	output/0
NOT	input/0	← →	output/1
	input/1	← →	output/0

Test Pattern Generation

Test pattern generation is the process of generating an appropriate (minimal) subset of all input patterns to stimulate the inputs of a circuit such that a desired percentage of detectable faults can be exercised and detected [Abramovici et al., 1992; Jha et al., 2003]. The process can be divided in two distinct phases: (1) derivation of a test and (2) application of a test. For (1), one must first select appropriate models for the circuit (gate or transistor level) and for faults; one must construct the test such that the output signal from a faulty circuit is different from that of a good circuit. This can be computationally very expensive, but one must remember that the process is done only once during the design stage. The generation of a test set can be obtained either by algorithmic methods (with or without heuristics), or by pseudo-random methods. On the other hand, for phase (2), a test is subsequently applied many times to each device and thus must be efficient both in space (storage requirements for the patterns) and in time. Often such a set is not minimal, as near minimality may be sufficient. The main considerations in evaluating a test set are: (a) the time needed to construct a minimal test set; (b) the size of the test pattern generator, i.e., the software or hardware module used to stimulate the circuit under test; (c) the size of the test set itself; (d) the time to load the test patterns; and (e) the equipment required (if external), or the BIST overhead (see Figure 4).

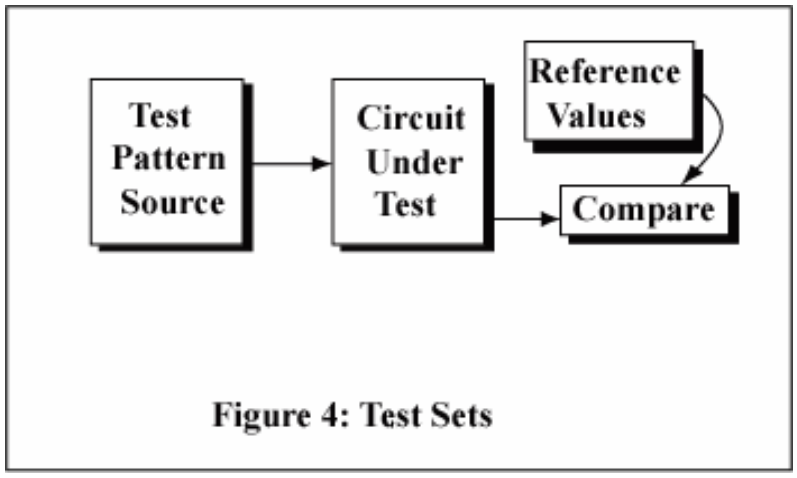

Figure 4: Test Sets

Most algorithmic test pattern generators are based on the concept of *sensitized paths,* through controllability and observability. Given a line in a circuit, one wants to find a sensitized path to stimulate a possible fault and carry its logical effect to an observable output. For example, to sensitize a path that passes through an AND gate, one must set all other inputs of the gate to logic 1 to permit the sensitized signal to carry forward.

Figure 5 summarizes the underlying principles of constructing a test pattern for each fault through path sensitization, which is not a full algorithm, but underlies the overall logic. In Part (a) one wants to find possible input patterns to test the fault on line *g/0*. One must set line *g* to *1* and might expect a faulty signal to be *0*, thus the notation *g:1/0*. In forward propagation, line *h* must be set to *0* in order for the fault to become observable at the output *Z: 1/0*. In Part (b) of Figure 5, the second phase of backward

propagation is shown, where possible controlling inputs are set. Lines *a* and *d* must be set to *1* to control line *g* being *1*, thus assigning *x1=x2=1*, which also forces line *f* to be *0*. This leaves *x3* with a choice of either *0* or *1* to maintain *h=0*. Thus 2 test patterns are found: *(x1 x2 x3) = {(1 1 0*) or (1 1 1)}.

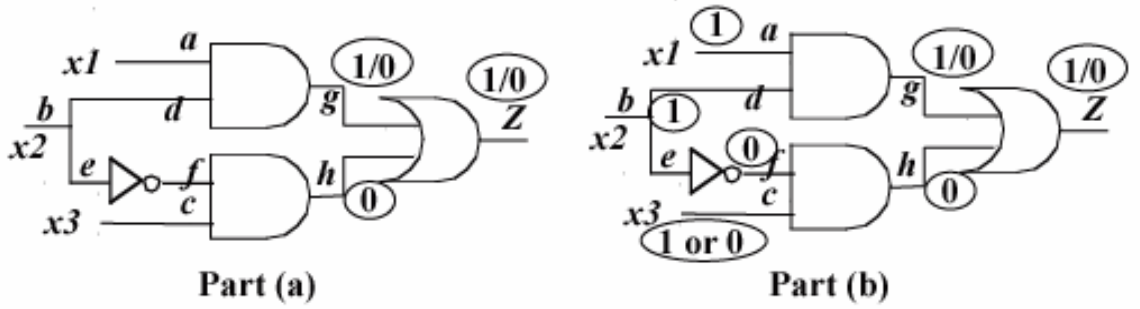

Figure 5: Path Sensitization

The best-known algorithms are the D-algorithm (precursor to all), PODEM, and FAN, plus many variations with heuristics and optimizations, often tailored to certain classes of devices [Abramovici, 1992; Jha et al., 2003]. Major steps can be identified in most automatic test pattern generation (ATPG) programs: (1) listing the signal on the line on which a fault should be detected; (2) sensitizing the path from that line to a primary output such that the fault can be observed; (3) determining the primary input conditions necessary to set the testing signal (back propagation); and (4) repeating this procedure until all detectable faults in a given fault set have been covered. Powerful heuristics are used to speed the steps by aiding in the sequential selection of faults to be examined and by cutting the amount of back and forward propagation necessary. The overall process is summarized graphically in Figure 6.

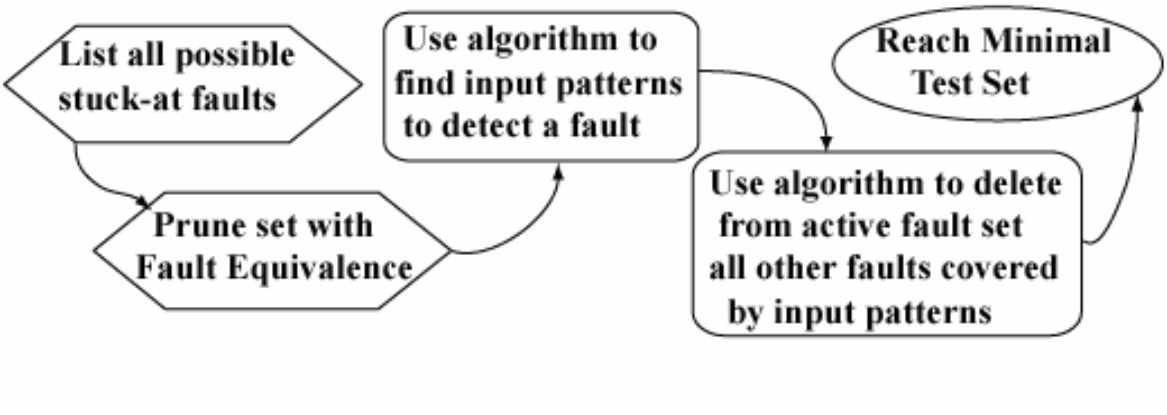

Figure 6: Process for test patterns generation

Notwithstanding heuristics, algorithmic test pattern generation is very computationally expensive and can encounter numerous difficulties, especially in certain types of networks. Newer alternatives are based on pseudo-random pattern generation [Bardell et al., 1987] and fault simulation. With this strategy, a large set of patterns is generated pseudo-randomly with the aid of an inexpensive (hardware or software) generator. Typical choices for these are linear feedback shift registers (LFSR) and linear cellular automata registers (LCA) (see below). The pseudo-random set is used to stimulate a circuit, and, using a fault simulator, one can evaluate the number of faults that are covered by this set. An algorithmic test pattern generator is then applied to find coverage for the remaining faults - hopefully, a small number - and the pseudo-random set is thus augmented. The disadvantages are that the resulting set is very large and fault simulation is also computationally expensive. However, this method presents an alternative for circuits where the application of deterministic algorithms for all faults is infeasible. Pseudo-random pattern generation is also the main alternative when BIST is introduced, as the overhead of storing even a minimal test set would be impractical for in circuit testing.

Output Response Analysis and Built-In Self-Test

Output response analysis encompasses methods which focus on the output stream, with the assumption that the circuit is stimulated by either an exhaustive or a pseudo-random set of input combinations. Especially when designing a circuit with BIST, one must decide how to check the correctness of the circuit's responses [Bardell et al., 1987]. It is infeasible to store on-chip all expected responses, and thus a common solution is to reduce the circuit responses to relatively short sequences: this process is called *data compaction or signature analysis,* and the short, compacted resulting sequence is called a *signature*. The normal configuration for data compaction testing is shown in Figure 7.

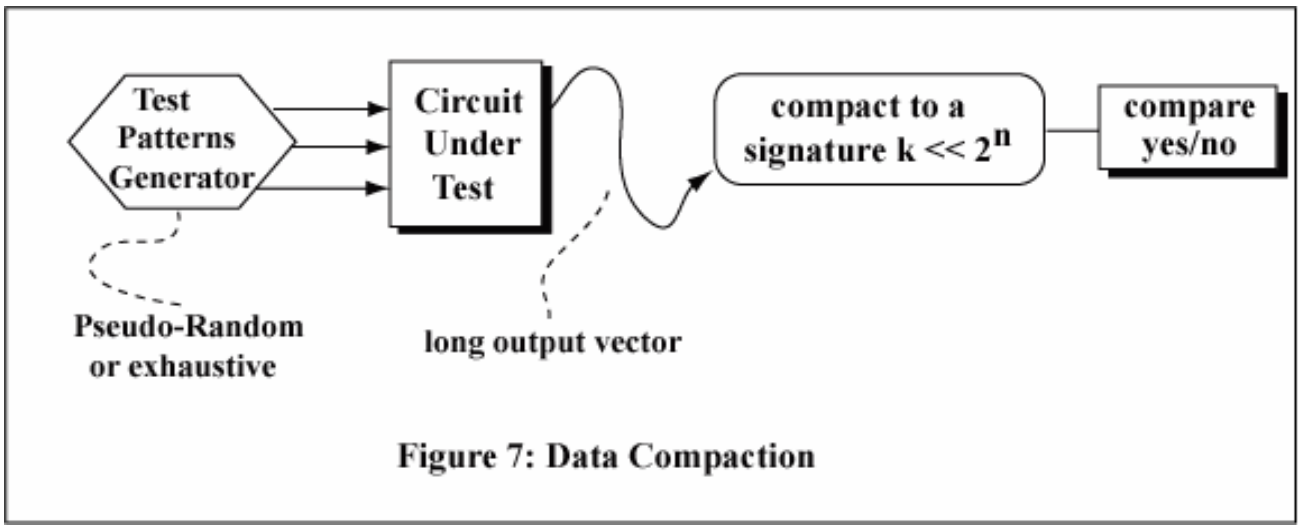

Figure 7: Data Compaction

The circuit is stimulated by an input pattern generator (pseudo-random or even exhaustive if $n < 20$); the resulting long output vector(s) is compacted to a very short signature of length k, where k is usually 16 to 32 bits. The signature is then compared to a

known good value. The main advantages of this method are that (1) the testing can be done at circuit speed by appropriate choice of the pseudo-random generator; (2) there is no need to generate algorithmic test patterns; and (3) the testing circuitry involves a very small area, especially if the circuit has been designed using scan techniques (see next Chapter).

The issues revolve around designing very efficient input generators and compactors. The main disadvantage of this method is the possibility of aliasing. When the short signature is formed, a loss of information occurs, and it can be the case that a faulty circuit produces the same signature as a fault-free circuit. The design method for data compaction aims at minimizing the probability of aliasing. Using the compactors described below, the probability of aliasing has been theoretically proven to be 2^{-k}, where k is the length of the compactor and thus the length of the signature. It is important to note that the result is asymptotically independent of the size and complexity of the circuit under test. For example, for $k = 16$, the probability of aliasing is about 10^{-6}. The empirical results show that in practice this method is even more effective. Most of all, this is the chosen methodology when BIST is required for its effectiveness, speed, and small area overhead.

A secondary issue in data compaction is in the determination of the expected "good" signature. The best way is to use fault-free simulation for both the circuit and the compactor, and then the appropriate comparator can be built as part of the testing circuitry [Bardell et al., 1987; Abramovici, 1992; Jha et al., 2003].

The most important issues are in the choices of a pseudo-random generator and a compactor. Although no "perfect" compactor can be found, several have been shown to be very effective. Several compaction techniques have been researched: counting techniques, as in one's count, syndrome testing, transition count, and Walsh spectra coefficients; and techniques based on linear feedback shift registers (LFSR) or linear cellular automata registers (LCA). Only these latter ones are discussed here. LFSRs and LCA are also the preferred implementation for the input pattern generators.

Pseudo-Random Pattern Generators

An Autonomous LFSR is a clocked synchronous shift register augmented with appropriate feedback taps and receiving no external input [Bardell et al., 1987]. It is an example of a general linear finite state machine, where the memory cells are simple D flip-flops and the next state operations are implemented by EXOR gates only. Figure 8(a) shows an example of an ALFSR of length $k = 3$. An ALFSR of length k can be described by a polynomial with binary coefficients of degree k, where the nonzero coefficients of the polynomial denote the positions of the respective feedback taps. In Figure 8(a), the high-order coefficient for x^3 is 1, and thus there is a feedback tap from the rightmost cell s_2 (reading from right to left); the coefficient for x^2 is 0, and thus no feedback tap exists after cell s_1; however, taps are present from cell s_0 and to the leftmost stage since x^1 and x^0 have nonzero coefficients. Since this is an Autonomous LFSR, there is no external input to the leftmost cell.

The state of the ALFSR is denoted by the binary state of its cells. In Figure 8(a), the next state of each cell is determined by the implementation given by its polynomial and can be summarized as follows: $s_0^+ = s_2$, $s_1^+ = s_0 \oplus s_2$ *and* $s_2^+ = s_1$, where the s_i^+ denote the next state of cell s_i at each clock cycle. If the ALFSR is initialized in a nonzero state, it cycles through a sequence of states and eventually comes back to the initial state, following the functionality of the next-state rules implemented by its polynomial description. If the polynomial chosen to describe the ALFSR is primitive, the ALFSR of length *k* cycles through all possible 2^{k-1} nonzero states in a single cycle (see the theory of Galois fields for the definition of primitive), and such polynomials can be found from tables [Bardell et al., 1987]. By connecting the output of each cell to each input of a circuit under test, the ALFSR provides an ideal input generator, as it is inexpensive in its implementation and it provides the stimuli in pseudo-random order.

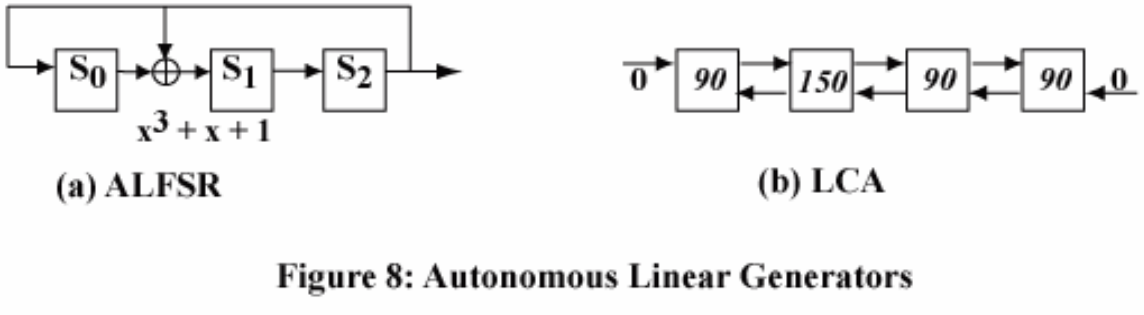

Figure 8: Autonomous Linear Generators

An alternative to an ALFSR is an LCA - a one-dimensional array composed of two types of cells: rule 150 and rule 90 cells [Cattell et al., 1996]. Each cell is composed of a flip-flop that saves the current state of the cell and an EXOR gate used to compute the next state of the cell. A *rule 150* cell computes its next state as the EXOR of its present state and of the states of its two (left and right) neighbors, as in $s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1}$, while a *rule 90* cell computes its next state as the EXOR of the states of its two neighbors only, as in $s_i^+ = s_{i-1} \oplus s_{i+1}$. As can be seen in Figure 8(b), all connections in an LCA are near-neighbor connections, thus saving routing area and delays. There are some advantages of using LCA instead of ALFSR: the localization of all connections, the ease of concatenation to obtain various sizes, and, and most importantly, it has been shown that LCA are much "better" pseudo-random pattern generators when used in autonomous mode, as they do not show the correlation of bits due to the shifting of the ALFSRs. Finally, the better pattern distribution provided by LCA as input stimuli has been shown to provide better detection for delay faults, requiring a two-patterns stimuli.

As for ALFSR, LCA are fully described by a characteristic polynomial, and through it any linear finite state machine can be built either as an ALFSR or as an LCA. It is, however, more difficult, given a polynomial, to derive the corresponding LCA, and

tables are now used. The main disadvantage of LCA is in the area overhead incurred by the extra EXOR gates necessary for the implementation of the cell rules.

Data Compaction or Signature Analysis

If the leftmost cell of an LFSR is connected to an external input, as shown in Figure 9, the LFSR can be used as a data compactor [Bardell et al., 1987]. In general, the underlying operation of such an LFSR is to compute polynomial division over a finite field, and the theoretical analysis of the effectiveness of signature analysis is based on this functionality, with the fundamentals coming from the theory of cyclic codes. The polynomial describing the LFSR implementation is seen to be the divisor polynomial. The binary input stream can be seen to represent the coefficients (high order first) of a dividend polynomial. For example, if the input stream is *1001011* (bits are input left to right in time), the dividend polynomial is represented as $x^6 + x^3 + x + 1$. After seven clock cycles for all the input bits to have entered the LFSR, the binary output stream exiting from the right denotes the quotient polynomial, while the last state of the cells in the LFSR denotes the remainder polynomial.

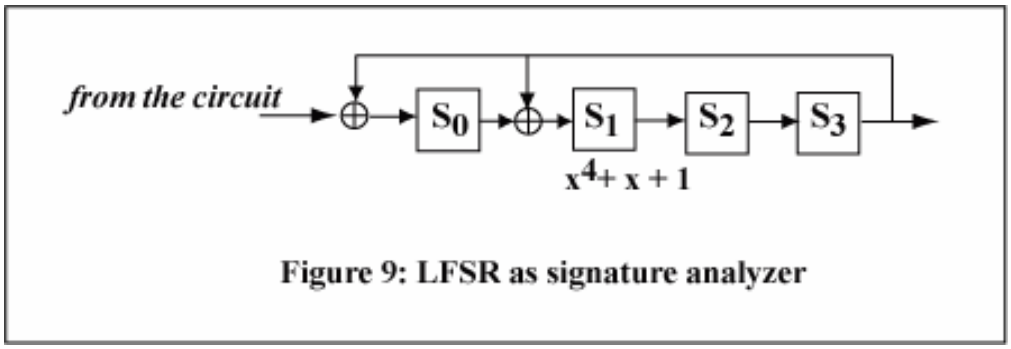

Figure 9: LFSR as signature analyzer

In the process of computing a signature for testing a circuit, the input stream to the LFSR used as a compactor is the output stream from the circuit under test. At the end of the testing cycles, only the last state of the LFSR is examined and considered to be the compacted signature of the circuit. In most real cases, circuits have many outputs, and the LFSR is converted into a multiple-input shift register (MISR). A MISR is constructed by adding EXOR gates to the input of some or all the flip-flop cells; the outputs of the circuit are then fed through these gates into the compactor. The probability of aliasing for a MISR is the same as that of an LFSR; however, some errors are missed due to cancellation. This is the case when an error in one output at time t is canceled by the EXOR operation with the error in another output at time $t + 1$. Assuming equal probabilities for the different errors occurring, the probability of error cancellation has been shown to be 2^{1-m-N}, where m is the number of outputs compacted and N is the length of the output streams.

Given that the normal length of signatures used varies between $k = 16$ and $k = 32$, the probability of aliasing is minimal and considered to be acceptable in practice. In MISR, the length of the compactor also depends on the number of outputs tested. If the number of outputs is greater than the length of the MISR, algorithms or heuristics exist

for combining outputs with EXOR trees before feeding them to the compactor. If the number of outputs is much smaller, various choices can be evaluated. The amount of aliasing that actually occurs in a particular circuit can be computed by full fault simulation, that is, by injecting each possible fault into a simulated circuit and computing the resulting signature. Changes in aliasing can be achieved by changing the polynomial used to define the compactor. It has been shown that primitive polynomials, essential for the generation of exhaustive input generators (see above), also possess better aliasing characteristics. It is also been shown that all implementations of such linear finite state compactors, be it LFSR or LCA, possess the same aliasing properties.

Summary

Accessibility to internal dense circuitry is becoming a greater problem, and thus it is essential that a designer consider how a device will be tested and incorporate structures in the design to assist that testing. Formal DFT techniques are concerned with providing access points for testing. As test pattern generation grows even more prohibitively expensive, probabilistic solutions based on data compaction and using fault simulation are becoming more widespread, especially if they are supported by DFT techniques and can avoid the major expense of dedicated external testers. However, any technique chosen must be incorporated within the framework of a powerful CAD system providing semiautomatic analysis and feedback.

Defining Terms

Aliasing: It occurs if the faulty output produces the same signature as a fault-free output.
Built-in self-test (BIST): The inclusion of on-chip circuitry to provide testing.
Fault coverage: The percentage of detected faults over all possible detectable faults.
Fault simulation: An empirical method used to determine how faults affect the operation of a circuit and/or also how much testing is required to obtain a desired fault coverage.
LFSR: A shift register formed by flip-flops and EXOR gates, chained together, with a synchronous clock, used either as input pattern generator or as signature analyzer.
MISR: Multiple-input LFSR.
Off-line testing: A testing process carried out while the tested circuit is not in use.
On-line testing: Concurrent testing to detect errors while circuit is in operation.
Pseudo-random pattern generator: It generates a binary sequence of patterns where the patterns appear to be random in the local sense, but they are deterministically repeatable.
Random testing: The process of testing using a set of pseudo-randomly generated patterns.
Signature analysis/data compaction: A test where the output responses of a device over time are compacted into a characteristic value called a *signature*, which is then compared to a known good one.
Stuck-at fault: A fault model represented by a signal stuck at a fixed logic value (0 or 1).
Test pattern (test vector): An input vector such that the faulty output is different from the fault-free output (the fault is stimulated and detected).

References

1. M. Abramovici, M.A. Breuer and A.D. Friedman, Digital Systems Testing and Testable Design, Rockville, Md.: IEEE Press, 1992.
2. P.H. Bardell, W.H. McAnney, and J. Savir, Built-In Test for VLSI: Pseudorandom Techniques, New York: John Wiley and Sons, 1987.
3. K. Cattell and J.C. Muzio, "Synthesis of one-dimensional linear hybrid cellular automata," IEEE Trans. Computer Aided Design, vol. 15, no. 3, pp. 325–335, 1996.
4. N. Jha and S. Gupta, Testing of Digital Systems, Cambridge University Press, 2003.
5. T.W. Williams (Ed.), VLSI Testing, Amsterdam: North-Holland, 1986.

Further Information

The author refers the reader to the books by Abramovici et al. [1992] and Jha et al. [2003] which, at the present time, give the most comprehensive view of testing methods and design for testability. More information on deterministic pattern generation can also be found in Fault Tolerant Computing, edited by D.K. Pradhan,. For newer approaches to random testing, the book by Bardell et al. [1987] contains basic information. The latest state-of-the-art research is to be found mainly in proceedings of the IEEE International Test Conference.