ELSEVIER

Journal of Systems Architecture 46 (2000) 721–747

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Testing and built-in self-test – A survey

Andreas Steininger *

Institut für Technische Informatik, Vienna University of Technology, Treitlstrasse 3/182-2, A-1040 Vienna, Austria

Received 27 August 1998; received in revised form 4 August 1999; accepted 6 September 1999

Abstract

As the density of VLSI circuits increases it becomes attractive to integrate dedicated test logic on a chip. This built-in self-test (BIST) approach not only offers economic benefits but also interesting technical opportunities with respect to hierarchical testing and the reuse of test logic during the application of the circuit.

Starting with an overview of test problems, test applications and terminology this survey reviews common test methods and analyzes the basic test procedure. The concept of BIST is introduced and discussed, BIST strategies for random logic as well as for structured logic are shown.

Keywords: Built-in self-test; Boundary-scan; Test pattern generation; Response analysis; Scan path

1. Introduction

The increasing functional complexity of electronic components and systems makes testing a challenging task, particularly under the constraints of high quality and low price. Considering that testing represents a key cost factor in the production process (a proportion of up to 70% of total product cost is reported in [1–3]), an optimal test strategy can be a substantial competitive advantage in a market comprising billions of electronic components and systems. It is therefore not surprising that the "National Technology Roadmap for Semiconductors" [4] of the United States views testing as a one of the six "Grand Challenges" for the semiconductor industry within the next 15 years.

This survey reviews common test methods and introduces the concept of built-in self-test (BIST) as a promising approach to overcome the limitations of the traditional test technology that become more and more conspicuous. Section 2 starts with the motivation for testing. Section 3 summarizes the fault models that are applied for testing, while test quality ratings are introduced in Section 4. Section 5 discusses the different phases during which a test is applied to a product. Basic test methods are presented in Section 6 and the procedure of a test is analyzed in Section 7. Concept and advantages of BIST are discussed in general in Section 8, while Section 9 presents the specific techniques for BIST of structured logic. The survey concludes with Section 10.

2. Motivation for testing

Testing has become an important issue in the production process of each electronic system,

* Tel.: +43-1-58801-18250.

E-mail address: steininger@vlsivie.tuwien.ac.at (A. Steininger).

1383-7621/00/$ - see front matter
PII: S1383-7621(99)00041-7

board or VLSI chip. *Design for test* is increasingly being preferred over tricky ad hoc design solutions, and all major electronic companies spend a considerable proportion of production cost and engineering resources for testing. The motivation for this becomes obvious from a more global point of view: Although testing incurs a lot of efforts it is finally an important means to reduce overall cost significantly. This is illustrated by the following examples:

- While the actual material cost is only a negligible proportion of the product value, the cost of repair increases by a factor of 10 with each production stage [5]. It is much cheaper to reject several dies than to locate and exchange a defective chip in a complete system. As a consequence no customer is willing to bear the risk of using defective components and therefore (a) only accepts suppliers that guarantee low defect rate and (b) often performs an incoming test for supplied parts. Low defect rate of the product can be guaranteed by extensive outgoing product tests only.
- VLSI chips have reached an enormous complexity and still their density doubles every two years [6]. This makes it impossible to rule out faults during design and production, even with the best design tools and fabrication processes available. However, short time-to-market is critical to profitability. If testing facilitates rapid diagnosis and thus provides a means to avoid fatal production delays resulting from excessive debug time or shipping defective products, it is worth the additional cost.

The so-called *factory testing* is the classic application area of testing. Another equally important motivation for testing comes from the area of dependable computing. The increasing level of integration results in small feature size, small charges in the storage elements and high proximity of functional units. This not only requires extreme care in the chip layout and manufacturing process, but also makes the circuits highly susceptible to external faults. The trend towards high clock frequency and low power consumption further aggravates the situation.

At the same time computers are increasingly used for safety-critical applications (e.g. in power plants, transportation systems and medicine), where a system failure may have disastrous consequences. Testing has been recognized as a valuable means to (a) check system installation and configuration after maintenance activities, (b) ensure correct system functionality at start-up, and (c) avoid masking and accumulation of errors during operation. Availability of a system (or of redundant system components) can be significantly increased, if testing is employed to allow rapid diagnosis after a failure. These facts clearly show the economic potential of an efficient test strategy in the rapidly growing area of dependable systems.

Although the above two fields of test application appear quite disjoint, it is technically and economically attractive to merge them. It will be shown that the concept of BIST provides an ideal starting point for a unified test approach for the complete life cycle of an electronic system.

3. Fault models for testing

3.1. Physical models

According to Amerasekara and Najm [7] failures in electronic components can be classified into three groups with respect to their origin (Table 1): electrical stress failures, intrinsic failure mechanisms and extrinsic failure mechanisms. Table 1 gives typical examples for each of these groups, for a detailed discussion see [7].

3.1.1. Electrical stress failures

Being an event dependent failure mechanism electrical stress is a continuous source of device defects over product lifetime. It is most often caused by improper handling.

Table 1
Global classification of component failures

Failure group	Relevant parameter	Time distribution of failures
Electr. stress	Handling	Continuous
Intrinsic	Technology	Predominantly infant but also wear-out
Extrinsic	Process	Yield loss
	Packaging	Wear-out, rarely infant
	Radiation	Continuous

3.1.2. Intrinsic failures

The group of intrinsic failures subsumes all crystal related defects. Since such defects depend very much on the maturity of the manufacturing process, they most often lead to yield loss or infant mortality, rarely to wear-out effects (gate-oxide wear-out and wear-out due to surface charge effects or ionic contamination have been observed). Type and manifestation of intrinsic failures are determined by technology: Gate-oxide defects are MOS-specific, while current gain shifts are a typical bipolar defect manifestation.

Performance degradation is a long-term effect of intrinsic failures.

3.1.3. Extrinsic failures

Extrinsic failures comprise all defects related to interconnection, passivation and packaging. They can be classified into three categories with respect to the time of defect manifestation:

- severe process deficiencies resulting in easy-to-detect errors (e.g. open bonding);
- wear-out effects affecting long term reliability (e.g. moisture-related failures);
- radiation-related errors continuously occurring over product life-time.

The probability of wear-out defects is strongly influenced by the package type. A considerable percentage of field failures due to packaging can be traced to moisture in the package. The widely used plastic package exhibits the worst quality.

Corrosion is affected by the relative humidity inside the package. For constant ambient humidity the package-internal relative humidity decreases with rising power dissipation of the device. That is why devices with low power dissipation such as CMOS devices operated at low frequencies are more susceptible to corrosion than other devices.

The order of importance of the failure mechanisms further depends on parameters like device size, maturity of the technology, and extent and effectiveness of the screening applied after production. With a proportion of 58% [7] electrical stress induced defects play an outstanding role in the field failures. Terrestrial cosmic rays have been reported to be a significant source of errors in both DRAMs and SRAMs. Alpha particles from materials on the chip have also been identified as cause for soft errors in DRAMs, with low supply voltage, "1" pattern and short access time leading to the highest sensitivity [8].

The vast majority of failure mechanisms is extremely temperature dependent: High temperature or temperature cycling leads to a significant increase in failure rate, the same applies for high supply voltage.

3.2. Logic models

Table 2 gives relations between the major physical defect mechanisms like migration, diffusion or gate oxide breakdown and their electrical manifestations. These manifestations are summarized in Table 3.

If these low-level electrical effects could be directly mapped to functional failures of the logic blocks of the device (AND, OR etc.), their impact on system operation could be easily determined. There are, however, no reasonable models that clearly establish such a relation [9]. An open circuit, e.g., may have various consequences depending on which specific role the affected part plays in the circuit. Solutions for specific problems can be found by simulation, of course.

The traditional *stuck-at* logic fault model therefore largely ignores the underlying physical effects and assumes two types of manifestation: One node (input or output) of a functional logic block is permanently fixed to a logic value by the defect. This value may either be logic 1 for the *stuck-at*-1 *fault* or logic 0 for the *stuck-at*-0 *fault*.

Stuck-at faults are a poor descriptor of actual defects, notably in CMOS [10], since many defects tend to cause *open faults*, i.e. faults causing one node of a functional logic block to be at an undefined logic value (e.g. as a result of an open circuit). It is therefore desirable to include open faults in the fault model. Open faults, however, tend to induce sequential behavior into a combinational circuit, which substantially complicates the testing strategy [11] (see Section 7.2.1). That is why the stuck-at model is still the most popular fault model for testing. Fortunately, it could be shown that tests generated for single stuck-at faults are also effective at detecting other types of faults.

Table 2
Classification of failure mechanisms in semiconductor devices

Failure group	Failure type	Failure mechanism	Typical reasons	Electr. effects
Electrical stress	Electrical over-stress (EOS)	Over-voltage or over-current with duration >1 μs (typ. 1 ms) melts silicon, metallization or bonding	Power supply transients, lightning surges, bad design	Short circuit, open circuit
	Electrostatic discharge (ESD)	Sudden (0.1 ns to 1 μs) discharge of electrostatic potentials of 100 V to 20 kV causes gate oxide breakdown	Improper handling, missing ESD protection	Short circuit, open circuit, + leakage current
	Latch-up	Parasitic pnpn device forms low resistance path between VCC and ground, melting effects like EOS	Voltage transient or current transient on power supply or I/O	Short circuit, open circuit
Intrinsic	Gate oxide wear-out	Time-dependent dielectric gate oxide breakdown due to weakness in the oxide film	Processing deficiency (degraded gate oxide integrity), high electric fields (e.g. erasing of programmable memories)	+ power dissip., speed decrease, + leakage current
	Ionic contamination	Mobile ions collect at the gate oxide boundary and induce extra charge	Processing deficiency (contaminated equipment, environment or package)	Threshold modif., − drive current, + leakage current
	Surface charge effects on isolation	Charge movement through isolation regions	Processing deficiency (oxide growth)	Threshold modif., + leakage current, + current gain
	Charge effects and I–V instability	Slow trapping of electrons in gate oxide	High electric fields (e.g. erasing of programmable memories)	Threshold modif., + leakage current
		Hot carriers create electron–hole pairs by impact ionization	High electric fields (e.g. erasing of programmable memories)	Parametric shifts
		Oxide damage by plasma related production steps	Processing deficiency (etching, resist strip, vapor deposition)	Threshold modif., + leakage current, − current gain
	Dislocations and crystalline defects	Structural defects in the silicon lattice cause leakage across reverse biased pn-junctions	Processing deficiency (silicon lattice defects, impurities)	Threshold modif., + leakage current, − current gain
	Piping	Shorts through current paths along dislocations	Processing deficiency (dislocations)	+ leakage current
Extrinsic (metallization)	Electromigration	High current densities force movement of metal atoms	Processing deficiency (inhomogeneities in the metal line), (DC) current stress conditions	Short-circuit, open circuit, + line resistance
	Contact migration	Interdiffusion of contact metal and silicon	(DC) current stress conditions	Open circuit, + contact resist. + leakage current
	Via migration	Migration of metal in multilevel metal systems	(DC) current stress conditions	Open circuit, + contact resist.
	Step coverage	metallization at topographic steps in the surface of the wafer tends to be thinner, which leads to high current	Processing deficiency (aspect ratio, step size), (DC) current stress conditions	Short-circuit, open circuit, + line resistance

Table 2 (*continued*)

	Microcracks	densities, electromigration and mechanic cracks	Processing deficiency (aspect ratio, step size)	Open circuit
	Stress induced migration	Mechanical (tensile) stress induces transfer of metal atoms	Processing deficiency (thermal mismatch between metal and passivating film)	Short-circuit, open circuit + line resistance
Extrinsic (package and assembly)	Die attach failures	Degraded heat dissipation or hot spots due to voids and cracks, corrosion due to introduction of contamination and moisture	Packaging deficiency (temperature, materials, velocity)	Parametric shifts
	Bonding failures	Bond looping (tension creating fracturing of the bond wire) bond lagging (shorts between adjacent bond wires) interdiffusion of metals, bond lifts, whisker growth, dendrite growth	Packaging deficiency (wire length, contamination, moisture, bonding pressure, materials, molding pressure, surface cleaning)	Short-circuit, open circuit, unstable contact + contact resist.
	Delamination, popcorn effect	expansion of moisture in the chip leads to delamination of lead frame and cracking of the die (popcorn)	packaging deficiency (moisture, thermal mismatch); temperature cycling, soldering	Open circuit unstable contact, + leakage current
	Corrosion	Degradation of metal properties due to chemical effects	Packaging deficiency (moisture, ionic catalysts)	Open circuit, + line resistance + contact resist., + leakage current
Extrinsic (radiation)	Soft errors	Bit flips due to external radiation	(terrestrial) cosmic rays	Bit flips
		Bit flips due to intrinsic radiation	Trace impurities of radioactive materials present in packaging	Bit flips

Table 3
Typical electrical fault manifestations

Contact problems	Open circuit High resistance of contact/line Unstable contact
Isolation problem	Short-circuit Low-resistance path
Parametric faults	Increased leakage currents Threshold shifts Change in current gain Increased power dissipation
Dynamic faults	Speed decrease
Storage faults	Bit flips

Further fault models on the logic level are conceivable (e.g. coupling of two logic values due to shorted signal lines) but not generally applied.

3.3. Functional models

On the next level of abstraction it can be investigated in which way the faulty functional block degrades the intended functionality of the device. While no model with general validity exists, specific solutions can be found based on the knowledge of the functionality of a given device. An example are RAMs that have a commonly agreed functionality, which facilitates the definition of generic functional RAM models (one such model is discussed in Section 9.1.1).

Table 4
Overview of functional defects in memories

Fault category	Fault type	Fault description
Stuck-at fault	Cell-stuck-at-0/1	
	Driver-stuck-at-0/1	
	Read/write line stuck-at-0/1	
	Chip-select line stuck-at-0/1	
	Data line stuck-at-0/1	
	Open in data line	
Coupling fault	Inversion coupling fault	Transition in cell A forces inversion of cell B
	Idempotent coupling fault	Transition in cell A forces value (0 or 1) in cell B
	AND bridging fault	Value of bridge AB is logic AND of shorted cells or lines A and B
	OR bridging fault	Value of bridge AB is logic OR of shorted cells or lines A and B
	State coupling fault	A certain state of cell A forces a value (0 or 1) in cell B
Neighborhood pattern sensitive fault (NPSF)	Active/dynamic NPSF	Change in cell N also changes cell B; N is in physical proximity of B
	Passive NPSF	Pattern in N inhibits changes in B; N is in physical proximity of B
k-coupling fault	(general)	Transition in cell A changes the value of cell B if another k - 2 cells are in a certain state
Transition fault		Cell B fails to transit from 1 to 0 or from 0 to 1
Address decoder fault	Address line stuck	
	Open in address line	
	Shorts between address lines	
	Open decoder	For a detailed description of address decoder faults see [12,13]
	Wrong access	
	Multiple access	
	Arbitrary address mapping	
Data retention fault		Due to a broken SRAM pull-up cell B enters state X (floating) instead of 1. Leakage finally causes a transition to 0 after e.g. 100 ms

Based on these RAM models a quite general fault model has been established for memories and is widely used (though with different level of detail and extent). Table 4 gives an overview of this model, (see for a detailed discussion [12]).

The stuck-at faults introduced in context with logic faults also appear in Table 4, however with a different meaning: On the logic level these faults are related to logic primitives irrespective of their contribution for device function, while on the functional level the fault is viewed from the point of device functionality. Hence only a subset of the logic stuck-at faults propagates as such to the functional level. As an example, the transition fault may be the result of a stuck-at-0 fault at the set or reset input of the cell flip–flop.

It has been found that the effects of faults in the address decoder are very exceptional, therefore they are treated separately [12,13], although some of them could as well be expressed as stuck-at faults, open faults or coupling faults. Faults in the read/write logic can be mapped into address decoder faults.

3.4. Parametric faults

Not every defect has direct consequences on the logic functionality of the device. The group termed

"parametric faults" in Table 3 is an example for this: A shift in the threshold of an input may or may not cause a malfunction depending on the analog input signal level actually applied. An even more subtle problem is increased power consumption which has no immediate negative effect on circuit functionality, but may be the cause of a failure in battery supplied applications.

Parametric faults are quite expensive to test for, since a parametric test unit with analog inputs and outputs is required. Moreover, the go/no go decision is not easy, if the parameter varies with temperature, humidity, power supply voltage, input currents or output loads. Unless the test is performed under worst-case conditions, the faulty parameter may be within limits. Parametric faults are often the consequence of bad design, but they may as well be an indication of a defect. Aging plays an important role in this context, since parametric faults tend to get worse over time.

In addition to the DC parameter faults listed in Table 3 there is an important class of faults concerning AC parameters like setup/hold-time, propagation delay, rise/fall-time or access time. These faults are often referred to as *dynamic faults* [14], although [12] defines dynamic faults as an extra class of time dependent chip internal faults "that can be detected by functional tests but not be directly measured at the component pin with special measuring equipment for delay". Dynamic faults are typically caused by global defects (too-thin polysilicon, too-thick gate-oxide etc.), while local defects tend to cause functional faults [12]. Many companies commonly test their chips for delay faults.

IDDQ-Testing, i.e. testing by monitoring the quiescent current of a device (see Section 6.3), is another prominent example of a parametric test.

4. Test quality ratings

The purpose of a test is to distinguish a circuit that will be performing as specified during operation from a defective one. In practice, part of the circuit functionality is not tested under the given conditions, which results in limitations on the completeness of a test. These limitations are indicated by the *fault coverage* of a test, which is defined as the percentage of faults covered by the test related to the total number of assumed faults. Central to this definition is a model of the faults to be detected, which means that fault coverage varies with the fault assumption. Consequently, a 100% fault coverage for the traditional stuck-at fault model may still be insufficient, since other types of fault (open faults, sequential faults) are not covered. Quality level defined as the percentage of defective circuits that pass the test, is therefore a function of total fault coverage, of which the common stuck-at model is simply an estimator [10].

A test is said to have high *diagnostic resolution* if it is not only capable of detecting but also of precisely locating and identifying a fault. While this is not a necessary property for a simple go/no go decision, it may be of vital importance for the identification of deficiencies in the manufacturing process.

The quality measures of a test must be balanced with cost factors like test duration, silicon area overhead for test circuitry (which, in turn, impacts chip yield), additional pins required for testing, efforts for test pattern generation, or a possible performance penalty of the circuit under test (CUT) due to the inserted test logic. Table 5 gives a summary of important test quality attributes [15].

Test cost depends very much on the desired quality level: A step from 97% to 99% fault coverage or the consideration of an additional fault type may increase test cost by an order of magnitude.

The qualitative relationship between test quality and test cost is given by the *testability* [16]: A circuit is highly testable if high test quality can be reached with low cost. Testability describes the suitability of a circuit's internal structure with respect to testing and is determined by two parameters [5]:

Controllability indicates how easily the system state can be controlled by the primary inputs (pins) of a circuit. Quantitative controllability ratings can be determined for each node in the circuit; most often an average over all nodes is given. In the example in Fig. 1 the controllability of node y is a measure of how easy it is to set the output of

Table 5
Important test quality attributes

Fault characteristics	Fault coverage	With respect to fault model
	Test scope	Chip internal circuitry
		Interchip wiring and I/O
		Board or system
	Fault types	Combinational faults
		Sequential faults
		Parametric faults
		Delay faults
Cost characteristics	Area overhead	Silicon overhead
		Interconnect overhead
	Pin overhead	Additional pins required for testing
	Performance penalty	Added path delays
	Yield decrease	Reduced production level (# of dies per wafer)
	Reliability reduction	Due to increased functionality and area
Other characteristics	Generality of solution	
	Test time	
	Diagnostic resolution	
	Required circuit modifications	

block A to a logical 1 or 0 by assigning values to the primary inputs of the circuit. It is a function of the controllability of nodes w, v and u.

Observability indicates how easily the system state can be observed at the primary outputs (pins) of a circuit. Again, the average of the observability ratings for each node is usually given. In Fig. 1 the observability of node y is a measure of how easy it is to propagate the value of node y to a primary output by assigning values to the unassigned primary inputs, and is a function of the controllability of node x and the observability of node z.

Numerous tools have been described in literature [16–19] that allow a quantitative computation

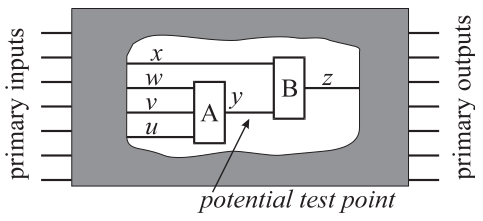


Fig. 1. Test point insertion to improve controllability and observability.

of testability. They use, however, different definitions and algorithms, so that the testability ratings cannot be compared.

Testability ratings are used to identify circuit areas where testing will become difficult. In such areas a redesign according to the rules of *design for testability* (DFT [5,20–24]) is mandatory. Redundant logic, for instance, cannot be tested, as it does not contribute to the device functionality. It must therefore be removed [25], if a fault coverage of 100% is aimed at.

It follows from the definitions of controllability and observability that testability can be substantially improved, if more nodes are made available directly at primary inputs and outputs. As a consequence *test points* are inserted in areas of low testability to allow direct observation or control of an otherwise internal node. Although heuristic approaches exist, optimal placement of test points is still an open issue [26,27].

In a conventional test approach each test point is connected to a dedicated pin, which results in a substantial increase of cost. In more recent designs test points are arranged as registers within a scan chain, so the number of required pins can be traded for an increase of test duration. Both

approaches ultimately sacrifice testability to reduce the number of test points.

It will be shown later that in a BIST environment the complete tester logic is located on the chip, therefore availability of internal nodes is by far less critical.

5. Tests applied during product life

Numerous tests with different purposes are applied to a circuit during its life, each having to meet specific requirements. In the following some typical examples will be discussed.

5.1. Factory test

During the production of a VLSI chip tests are applied at different packaging levels, e.g. wafer level, die level (rarely) or package level. This ensures early detection of defects and keeps the cost of a defective product low. Probing is one of the most difficult problems for these tests in conventional approaches. If a feedback to the fabrication process is desired, diagnostic capabilities of the test are necessary. Diagnosis is also required if yield shall be improved by replacing defective parts of the circuit with redundant ones provided on-chip (e.g. in large memories [28]).

On the next packaging level an assembly test is performed to detect defective parts on a PCB or defective PCB traces. These tests are often based on a boundary scan (see Section 8.2), in special cases *flying probes* (probes that move from one contact to another with the sequence of contact coordinates being controlled by an external tester [29,30]) are employed. Again, diagnostic capabilities are desirable to allow fast defect location.

The test of complete subsystems and systems consisting of numerous PCBs is a challenging task that becomes more important as system complexity grows. IEEE has defined the so-called MTM bus for this purpose (IEEE 1149.5 Module Test and Maintenance Bus [31–33]).

Within the economic limits high fault coverage is the crucial requirement for all the above tests, while test duration must be reasonable but is not particularly critical. Many of these tests are also performed for incoming and outgoing products in context with quality management.

5.2. Start-up test

Testing is an essential means to ensure that the system begins operation in a fault-free condition. This is particularly important in a dependable system, since, e.g., a TMR (triple modular redundant) system starting operation with one faulty component is less reliable than its simplex version [5].

Two types of start-up can be distinguished in the following:

5.2.1. Cold start

A cold start is performed when the system is put into operation after a shutdown and possibly some period of inactivity. The following requirements must be balanced for the cold start test:

- High fault coverage is essential, though with a relaxed fault model, since production-specific faults need not be accounted for at this stage.
- Diagnosis is highly desirable to locate defective components for replacement.
- Usually test duration is not particularly critical.

5.2.2. Warm start

A warm start is performed to initialize the system or part of it. Initialization may be necessary to reintegrate a computing node that has been halted after error detection. A test of such a node is essential since the detected error may be of permanent nature.

The most stringent requirement on the warm start test concerns duration. If a component in a duplicated high availability system [34] fails, system operation is continued without redundancy until reintegration of the failed component. The risk of spare exhaustion resulting from excessive test duration must be balanced with the risk of reintegrating an erroneous component because of insufficient fault coverage.

If only a part of the system performs a warm start, it is crucial that the test does not interfere with the rest of the system that might continue operation. This requirement considerably complicates the test of interfaces.

5.3. On-line test

The tests discussed so far are applied off-line. As opposed to that on-line tests are applied while the system is performing its assigned task [35]. Two types of on-line test are distinguished, but may be effectively combined [6,36]:

5.3.1. Periodic test

Normal system operation is interrupted regularly for short periods during which a test is performed. These test periods may be aligned with phases of system inactivity or the test may be executed as regular system task. In any case the periodic test is performed only virtually in parallel to system operation. For this reason periodic testing is sometimes classified as off-line technique in literature [6].

5.3.2. Concurrent test

Concurrent testing is based on additional information on the system operation that is checked by additional hardware actually in parallel to normal system operation. Examples are:

- additional bits used for coding data such that the result of an arithmetic operation can be checked (*arithmetic codes* [37]);
- information on program structure stored in a circuit that monitors instruction flow (e.g. [38]);
- read-back of outputs to check the signal path (*wrap-around-system*).

A concurrent test is capable of directly detecting transient errors, which indicates how indeterminate the borderline between concurrent testing and error detection is.

On-line testing appears quite risky, since the following effects lead to higher error probability:

1. Every hardware addition made for testing is a new potential target of a fault.
2. Testing causes additional system activity, which may activate transient faults that would have remained ineffective otherwise.
3. By exercising rarely used or completely unused resources, testing may activate errors that may have remained latent otherwise.
4. An on-line test interferes with system operation. This interference complicates the execution of the assigned system task.

In spite of all these problems substantial improvements in overall system dependability can be achieved with on-line testing:

(a) On-line testing reduces the probability of multiple faults. Although the activation (and detection) of latent errors stated in (3) appears like a drawback at the first glance, it is a vital requirement for avoiding error accumulation. The majority of fault tolerant systems is designed to tolerate single errors, while multiple errors may lead to a failure. For this reason a periodic on-line test is mandatory in safety related applications [35,39]. The period between the tests is (roughly) determined by the mean time to failure (MTTF, the reciprocal value of the failure rate λ as defined in the MIL handbook [40]) of the circuit and the desired level of reliability.

(b) On-line testing complements error detection. Permanent faults can be reliably detected by a test, however, this type of fault is easily covered by standard error detection mechanisms, too. Concurrent tests also cover transient faults, while periodic tests do not: A transient fault occurring during a periodic test may be taken as an indication for adverse environmental conditions [13], but, since it may have affected the test procedure only, it cannot directly be correlated with erroneous execution of the assigned system task.

5.4. Maintenance test and diagnosis

Maintenance tests are performed to check proper system functionality either regularly or after upgrades of hardware/software/firmware components. Such tests often require the interruption of normal system operation.

In case of a permanent system failure efficient diagnosis is required to locate and repair the defect quickly, since system downtime may be very expensive.

6. Test methods

Based on the variety of models that exist for the description of a circuit on different levels of ab-

straction different test approaches have been developed. The most important ones are functional testing, scan test (also known as structural testing) and, for conventional CMOS devices, IDDQ test.

All of these tests have their particular strengths and limitations. Studies in [10,41] have demonstrated that no method had 100% coverage for the physical defects in practical examples, but that all were complementary to a large extent.

6.1. Functional testing

A very intuitive test approach is to *directly demonstrate that the specified function of the circuit* (*as a black box*) *is performed*. Examples are memory test by writing and reading or communication interface test by loopback. An exhaustive functional test is only viable for blocks with very limited functionality, therefore it is usually applied to regular structures like memories or Programmable Logic Arrays (PLAs) [9,12]. Since functional testing becomes prohibitively difficult and does not achieve sufficient coverage for circuits with complex functionality (with many inputs and/or many states like a processor [42] or state machine [43]) other test methods must be employed.

The functional test is based on the circuit specification alone, no information on the internal structure is required. While this is an advantage for testing off-the-shelf components or cores, it makes fault isolation and diagnosis difficult. If performed at-speed, functional tests are also capable of covering dynamic faults (e.g. hazards).

6.2. Structural testing, scan test

The structural test is a formal approach in which the *circuit is modeled as a collective of logic primitives* (*gates*, *flip–flops*) *which are tested*. The test itself is implementation dependent, but the test patterns required for each of the logic primitives are easy to determine. In a VLSI circuit the logic primitives cannot be accessed from outside, therefore provisions for testing must be made during the design. By far the most common approach is the *scan test*: All register cells of the circuit are combined to a shift register chain (*scan-chain*) in a special test mode. This partitions the circuit into a set of combinational subcircuits whose inputs and outputs are connected to the scan chain (see Fig. 2). All registers in the scan chain are *scan controllable* and *scan observable*, which turns them into *pseudo-primary inputs* and *pseudo-primary outputs*, respectively. This allows a complete test of the combinational logic and of the register cells in the scan chain as such. The connections of the register cells, however, cannot be completely tested, since they are changed during the test.

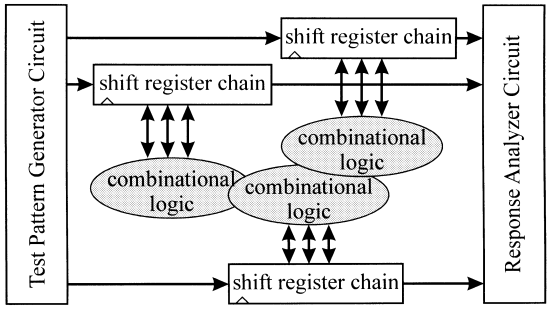


Fig. 2. Principle of the scan test.

Test duration can be reduced if several short register chains are formed instead of one long chain. However, the efforts for test pattern generation and response analysis are higher. The determination and generation of the test patterns is the most difficult part of the scan test. The respective methods are discussed in Section 7.2.

Scan design rules impose restrictions on the design, increase cost and sometimes tend to create long critical paths, therefore it may be advantageous not to use the *full scan* method described above, but a *partial scan*: The shift register chain does not comprise all register cells. Consequently the remaining logic is not purely combinational but contains sequential blocks (though with reduced complexity). Partial scan may therefore involve the application of several vectors and thus take several clock cycles for each targeted fault (see Section 7.3). Partial scan saves silicon area (non-scan latches require less space than scan latches) and routing resources (shorter scan path). The non-scan elements, however, must be predictably brought to any required state through

sequential operation to keep the circuit sufficiently controllable.

Today's CAD-tools can automatically convert an existing design into a scan design and generate the test patterns.

6.3. IDDQ-testing

Primary sources of failures in CMOS devices are short circuit defects (bridging defects) and open circuit defects. Since short circuit defects cause abnormal current flow, they can be detected by *monitoring the quiescent power supply current of the device*, which is normally caused by leakage currents and is in the order of 10^{-8} A. The abnormal currents introduced by defects are typically one or more magnitudes larger. During this IDDQ test different test vectors must be applied to activate state-dependent faults.

The IDDQ test is very attractive for CMOS RAMs since it

(a) is the single most sensitive test method [44] detecting all stuck-at faults with fewer test vectors than the functional test;

(b) complements the functional tests very effectively [10,45–47] by uniquely detecting gate-oxide shorts, defective p-n junctions, parasitic transistor leakage and even those stuck-at faults termed as undetectable in the context of functional stuck-at testing [48]; and

(c) can be used for the on-line detection of soft errors induced by so-called single event upsets (SEUs) such as heavy ions hitting a register cell [49].

However, measurement of the extremely low currents requires large settling times, which results in a slow measurement process. The choice of the IDDQ test limit and the test vectors is extremely critical [44,47]: Depending on these parameters the proportion of devices failing the test may vary in an interval as large as [1–100%]. IDDQ testing should be performed with the highest permissible supply voltage, since some effects may escape IDDQ detection for lower voltages. It has been shown that devices that fail IDDQ test but pass all other tests may still be functional but (a) tend to have reduced reliability (in terms of noise margins, failure rate etc. [44,48]) and (b) obviously have increased quiescent current.

IDDQ testing is losing its popularity since it will not work for advanced technologies, for which there is not a large difference in current levels between good and faulty devices. Instead, delay fault testing has gained high importance.

7. Basic test procedure

7.1. Test architecture

As shown in Fig. 3 the test of an electronic circuit is a stimulus/response measurement: In the first step a test pattern is applied to the circuit to bring it to a defined initial state or exercise some functionality. In the second step the test pattern is processed by the circuit, and in the third step the circuit's response is checked. This test procedure is repeated for different test patterns by a test controller.

7.2. Test pattern generation

One problem central to testing is the determination of an optimal sequence of test patterns under the following essential requirements:

- detection of (ideally) all defects assumed in the fault model,
- ease of generation/storage (low overhead),
- compactness (short test duration).

Extensive discussions on test pattern generation and the associated problems can be found in the literature [9,15,50–54]. The rest of this section gives a brief overview of the principles.

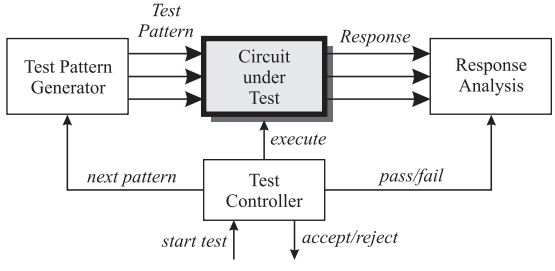


Fig. 3. Architecture of a typical test setup.

7.2.1. Combinational versus sequential logic

The function of a combinational circuit, i.e. a circuit without storage elements, is completely described by its truth table. A combinational fault (a fault in a time-invariant logic truth table) can be detected with a single test stimulus. A straightforward check of the truth table requires 2^n test patterns for a combinational circuit with n inputs. With a more sophisticated strategy the number of patterns can often be substantially reduced.

For the test of a combinational circuit an algorithmic solution exists (D-algorithm [55]) that finds all detectable faults. While the runtime of this exhaustive algorithm may be prohibitive in complex applications, more recent techniques attempt to reduce test generation time or test set size [50,56–58]. The Path Oriented Decision Making (PODEM) [59] algorithm is the basis for most of these modern test generation programs.

If the circuit under test, however, contains *sequential logic* (flip–flops and other digital storage elements), the test becomes much more complicated. Test patterns must be applied in different states; however, changing from one state into another may require a number of intermediate steps and test patterns. If a global reset of the circuit is not available, an initialization sequence or state identification is necessary, which makes the situation even worse [43].

Testing of sequential circuits is still a largely unsolved problem. This is particularly displeasing, since faults are more often found in sequential blocks than in combinational ones [60]. A smart approach is the scan test described in Section 6.2: The system state can be directly set by shifting the input pattern into the scan chain, while the remaining logic is of combinational nature. This converts the problem of testing a sequential circuit into that of testing a combinational one.

It has been shown [13] that open faults in CMOS circuits tend to behave sequential. This has the following reason: Due to the parasitic input capacitance the faulty input retains one logic state for an undetermined period. A state change is only caused by parasitic effects like cross coupling (i.e. electromagnetic interference from a neighboring signal) or leakage currents. This is a very severe effect, since such types of faults turn a combinational circuit into a sequential one and may, therefore, escape detection if a combinational test is applied only.

7.2.2. Divide and conquer

The above discussion has indicated that the number of required test patterns rises (roughly) exponential with the number of inputs even for a combinational circuit. Partitioning is therefore a vital means of reducing test pattern length [61,62]. If a combinational circuit with 20 inputs can be partitioned into two sub-circuits with 10 inputs for testing, an upper limit of only 2048 test patterns can be achieved instead of 1048 576. For CUTs with multiple outputs one simple means of partitioning is testing each output function separately, which usually involves only a subset of circuit inputs [15].

Partitioning is automatically achieved by the scan test to some extent and can be enhanced by the introduction of test points (as outlined in Section 4). For a very complex, unstructured circuit, however, the efforts for partitioning may be prohibitive. For this reason structuring is one important rule of design for testability.

7.2.3. Type of test sequence: Deterministic versus pseudo-random testing

The determination of a test pattern usually involves feedback from a fault model: In a first step a list of all faults considered by the fault model is made. In the second step a test pattern is assumed and all faults detected by it are removed from the fault list. Repeating this step for new test patterns progressively reduces the fault list. Towards the end of the iteration process the contribution of a new pattern decreases, since the fault list becomes small. One new pattern may be needed to remove one single fault from the list, while other new patterns do not make any contribution at all. Although an exhaustive application of this *deterministic algorithm* (D-algorithm [55]) promises the detection of all detectable faults, the duration of the search process and the length of the resulting test pattern may be excessive. A balance between coverage and cost must be found at this point.

A special case of the deterministic test is the *exhaustive test*, for which *all* possible test patterns are applied. While this test produces the best

possible coverage, it is impracticable for a complete VLSI chip. Considering the partitioning method described above a sum of exhaustive tests can be applied progressively to all parts of the circuit. This method is called *pseudo-exhaustive test* [5,15,56]. Test pattern generation for the exhaustive and pseudo-exhaustive test is trivial and does not involve fault simulation.

In a completely different approach a fixed number of test patterns is generated without feedback from the fault model. The sequence of test patterns is called *pseudo-random*, because it has some important properties of a random sequence, while being totally predictable and repeatable [5]. The coverage of the test sequence is checked by fault simulation (e.g. the fault list described above). This process of analyzing coverage via a fault simulation is called *fault grading* [63]. If a sufficient coverage level has been reached, the set is accepted, otherwise a new one is generated. One drawback of this *pseudo-random algorithm* is that the required length of the test sequence is often hard to determine a priori. Statistic frameworks have been derived for this purpose [5,56]; moreover the testability ratings discussed in Section 4 can be used as an indication. Although a sequence found by this approach does not necessarily have the best fault coverage possible, it has advantages over the deterministic search:

- For a reasonable coverage limit the determination of a sequence is extremely fast.
- The random patterns can easily be generated on-chip by a hardware random generator (Section 7.2.4).
- Random patterns have been shown to have much better coverage for nontarget defects, i.e. faults that are not considered by the fault model [26,64].
- Since cost is an extremely critical issue in every design, the deterministic approach may have to be terminated at a quite early iteration stage. In this case the results of the random approach with comparable cost may be superior.

It is quite usual to mix both algorithms in one of the following ways:

⇒ The deterministic algorithm is started on top of a number of random patterns. The addition of few deterministic test patterns guarantees detection of dedicated critical faults and improves coverage in general.

⇒ In an approach called *pseudo-deterministic testing* [65] a random sequence is selected such that it includes a number of dedicated test patterns that have been determined a priori by the deterministic algorithm (for hard-to-detect faults). There are also analytic methods of constructing a random generator such that its output sequence will include the required patterns [66]. In some cases it will be advantageous to draw patterns only from certain intervals of the complete sequence of patterns producible by an LFSR. This can be accomplished by reseeding the LFSR [67,68].

⇒ Alternatively, weights for the 1/0 probabilities can be defined for every bit position of special pseudo-random generator circuits [9]. These weights can be used to tune the test sequence such that the detection probability of hard-to-detect faults is increased. As an example, testing an AND gate requires more logic 1's to activate a fault, while testing an OR gate requires a high proportion of 0's at the inputs.

7.2.4. Test pattern generator circuits

The methods for test pattern generation are highly correlated with the types of test pattern discussed above. In the following some common approaches are discussed:

- *ROM* (*stored pattern test*): Since no restrictions on the sequence of patterns apply, this method can provide excellent fault coverage and is used in combination with the conventional deterministic algorithm. However, sequence length is directly proportional to ROM size, which is not so much a problem for external testers, but usually results in a prohibitive area overhead for built-in test.
- *Processor* (*test pattern calculation*): If the CUT includes a processor and memory, a test program can be employed to calculate an appropriate sequence of test patterns.
- *Counter* (*exhaustive test*): This simple way of test pattern generation is used for exhaustive and pseudo-exhaustive testing. It may also be applied as address generator for some memory test strategies (see Section 9.1)

- *Pseudo-random generator* (*random test*): These methods of recursive generation of test patterns provide reasonable coverage with low hardware-overhead. For this reason random testing is most often found in built-in test designs. Hardware circuits suitable as random generators are register chains like linear feedback shift registers (LFSRs [5,9,56]), cellular automata [69,70] or built-in logic block observers (BILBOs [5,56,71]). An LFSR modified such that it cycles through all possible states can be used as cheaper replacement of a counter for exhaustive testing, if the sequence of pattern is irrelevant (combinational testing). As mentioned above, pseudo-random pattern generators may be designed under the constraint that their output sequence includes a set of deterministically generated test vectors.

The stored pattern test method is classified as *off-line test pattern generation*, while processor, counter and pseudo-random generator are *concurrent test pattern generation methods* (test patterns are calculated while they are being applied) [15].

7.3. Test pattern application

In a scan design the clock is required for (a) scanning the test pattern in and out and (b) test pattern application. In a full scan architecture one clock cycle is required to apply the single test pattern while for partial scan the sequential logic remaining in the (original) CUT may have to be clocked several times: A specific sequence of test patterns must be applied to bring it to the desired state and test for the target fault [72]. With respect to the scanning method two approaches can be distinguished that will be described in the following: test-per-scan and test-per-clock.

7.3.1. Test-per-scan

Once the test pattern has been generated it is applied to the CUT input. For the usual synchronous circuits this means the test pattern is shifted in through the scan chain and the CUT is clocked. Finally the CUT output pattern is latched and shifted out via the scan chain for further analysis. The duration of this test-per-scan strategy (Fig. 4) is quite high: scanning out the previous

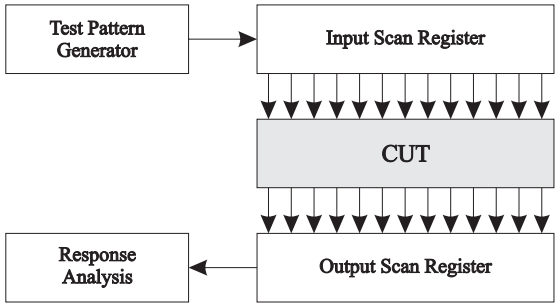


Fig. 4. Basic test-per-scan architecture.

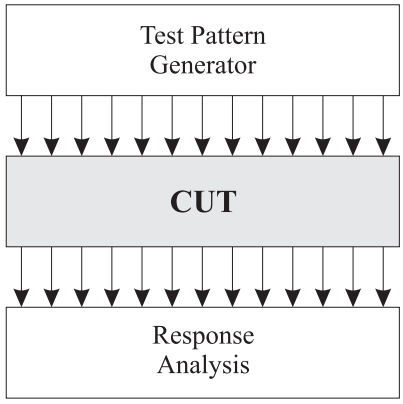


Fig. 5. Basic test-per-clock architecture.

(and simultaneously scanning in the next) test pattern requires n clock cycles for an n-stage scan path, which adds to the duration of the test pattern processing by the CUT.

7.3.2. Test-per-clock

Test duration can be significantly reduced, if one test pattern can be applied and processed with each clock cycle. In this test-per-clock strategy (Fig. 5) the test pattern generator produces an n-bit output word that is directly applied to the CUT (without scan). Similarly, the response analyzer is provided with an n-bit parallel input. This method requires direct access to every register in the chain therefore it is applicable only for BIST.

The drawback of test-per-scan is the additional hardware for the n-bit test pattern generator and the n-bit response analyzer that costs more than

the scan register in the test-per-scan architecture [9].

A special variation of the test-per-clock architecture is the *circular BIST* [73,74]: The response of the current test cycle is used to derive the next test pattern, either by a one-to-one correspondence or e.g. by shifting the whole word by one bit position. The response is shifted out for analysis only after a number of test cycles. In this approach hardware can be saved, because the combinational CUT itself is used as a feedback. However, since this feedback is nonlinear, coverage prediction and optimization are extremely difficult.

7.3.3. Scan path implementation

It is a basic requirement of the scan approach that it must be possible to clock data into system flip–flops from two different sources (see Fig. 6): One source is given by the normal signal flow during system operation (path "n"), the other is required to form the scan chain during test (path "s"). Multiplexers or two-port flip–flops are most often used for this purpose.

Several standard techniques for the implementation of the scan path are suggested in the literature [5,15,24,56,75]. Some of them are based on flip–flops, others assume latches in the scan path. The main issue in all approaches is to avoid race conditions that might be induced by clock skew in test mode. The flip–flop based approaches must employ two-edge clocking schemes, while the latch based approaches require a non-overlapping master/slave clock [5,56]. A widely used latch based design method is the Level Sensitive Scan Design, LSSD, first introduced in [76]. Other latch based examples are Random Access Scan [77] and Scan/Set Logic [78].

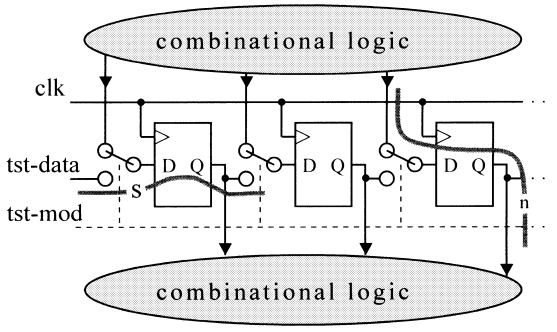


Fig. 6. Principle of the scan path implementation.

In some BIST implementations a multiplexer is inserted in the clock path to (a) allow a selection from different clock sources like e.g. system clock or a specific test clock (unless it is possible to use one single clock source for both test and application [79]) and (b) compensate for potential delays caused by multiplexers inserted for testing purposes in the data path. Such interference with system timing is extremely critical and therefore avoided whenever possible. It is a main source of performance degradation caused by test logic insertion.

7.4. Response analysis

To come to a decision whether the CUT is functioning properly the response bitstream (response vectors) must be compared with a known correct one. Such a reference response is usually derived by simulation, but may also be based on the observation of a known good device. Obviously, the most satisfactory solution is a direct comparison of the complete response data with a reference. This, however, implies knowledge of a substantial amount of reference data, which is possible only in specific cases:

- The complete reference data set can be stored in a ROM. Similar to the stored pattern approach for test pattern generation described in Section 7.2.4 this is a viable solution for external testing, but results in a huge hardware overhead, if applied for on-chip testers.
- A reference unit can be employed as a reference response generator. This approach, however, requires complete duplication of the CUT. In circuits for safety related applications redundant functional units might be available that can be employed as reference response generator. (Assuming a fault-free design we may neglect the probability of a common failure of reference generator and CUT.)
- A response generator circuit (state machine) can be constructed that is cheaper than the duplicated CUT but produces exactly the required data

sequence. This technique of analytically generating the complete response data set is called *response data compression*. It is not generally possible to find such a compression circuit.

Since the above methods are rarely applicable for BIST, other, cheaper solutions are commonly used in spite of their inferior performance. Their common principle is *response data compaction*: The original response data stream is reduced to a significant property (*signature*) by methods like

- *Transition counting*: The number of transitions contained in the data stream is counted.
- *Cyclic coding* (signature analysis): The data stream is fed through an LFSR whose final contents are used as a signature. A special case of this polynomial division method is parity (1 bit LFSR) which has been found to be the fastest method but also the most expensive and poorest in fault coverage [15]. The use of cellular automata instead of LFSRs is suggested in literature [9], but their usefulness is not yet fully established.
- *Ones counting*: The number of 1s in the data set is counted.
- *n-bit binary sum*: all words (width n) in the data stream are added, the sum is regarded as signature. This method is advantageous, if an accumulator is already present in the circuit [80].
- *Syndrome counting* [81]: If an exhaustive test pattern is applied to a combinational circuit, the output bitstream represents the output column of the circuit's truth table. The number of ones (*syndromes*) in this bitstream is used as signature. It has been shown that this technique is capable of detecting any single fault in the circuit, some modifications in the CUT may though be required.
- *Walsh spectra*: The Walsh spectrum of a circuit [82] can be used as a signature.

The signature (typically one single word) is then compared to a reference. Obviously storage of the reference signature is possible with significantly lower hardware overhead. However, at the same time the risk of error masking (*aliasing*) is introduced: Since the signature contains less information than the original data set, certain erroneous response data sets that have the same significant property as the correct one will also match the reference (one in 2^n for an n bit signature). Aliasing probability in general increases with the *reduction gain*, defined as the number of bits of the signature related to the size of the complete data set. Numerous attempts have been made to reduce aliasing probability by methods like multiple signatures [83,84], output data modification [85,86], or rearranging test vectors [87].

Compaction is commonly used to transform the sequence of response vectors into a single vector (*time compaction*). In some cases it is also applied to reduce vector width (*space compaction*, e.g. parity bit). A typical space compaction circuit is the multiple input shift register (MISR [5]).

8. The concept of BIST

8.1. Principle

According to a definition given in [9] "BIST is a design-for-test technique in which testing is accomplished through built-in hardware features". For chips with 1 million transistors and more the hardware required for testing can be integrated on the chip by dedicating a negligible percentage of the silicon (a very optimistic estimation of 3% is given in [5]) for the BIST logic. Such test hardware is implemented in concurrence with the "actual" design. In this way BIST represents an important step towards regarding the test as one of the system functions.

The principle of BIST is shown in Fig. 7: A BIST Controller generates the test patterns, controls the CUT clock and collects and analyzes the responses. This makes the external test interface

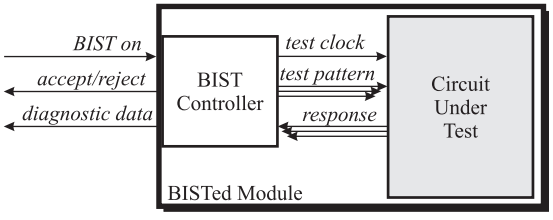


Fig. 7. Principle of built-in self-test.

much more compact than in the conventional test approach: The self test can be initiated over a single pin, the result (“accept” or “reject”) can be signaled via a second pin. Optionally, a serial bit stream with diagnostic data may be provided at a third pin.

8.2. Test access port as BIST interface

It has become popular to provide VLSI chips with a *JTAG boundary-scan* [88–90]. In a test mode this feature enables indirect access to each pin of the chip via a scan chain formed of dedicated boundary-scan registers. The virtual probe points that boundary scan adds to the I/O structures make it possible to exercise the internal logic without involving the I/Os (internal test) and the I/Os without involving the internal logic (external interconnection test). The Test Access Port (TAP, Fig. 8) controls scan operation and forms the standardized interface to the scan chain. This interface comprises only five signals:

– test clock input TCK;
– test mode select TMS;
– test data input TDI;
– test data output TDO;
– test reset input TRST (optional).

The TAP is a quite slow interface, but its specification supports many powerful instructions, including the optional RUNBIST command. Since only minimal data traffic is involved, this instruction establishes a sufficient communication between TAP and BIST controller, eliminating the need to reserve extra pins for a BIST interface [91,92].

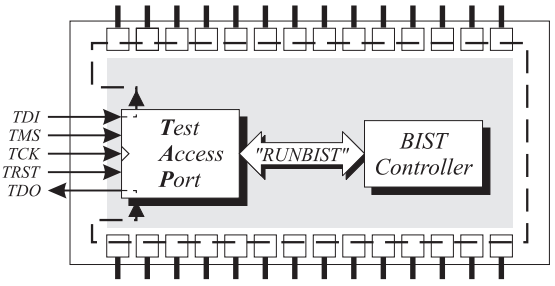


Fig. 8. Test Access Port for boundary scan.

8.3. Advantages of BIST over conventional testing

The automated test equipment (ATE) required for the conventional factory test of VLSI circuits usually includes expensive test hardware and probing solutions. This makes every second a device spends on the tester extremely costly. Most often the highly sophisticated test equipment cannot be reused for higher level tests or during other test phases than factory test. As opposed to that, BIST-logic designed for a specific VLSI circuit can be extremely helpful for other test purposes like maintenance and diagnosis or start-up test. The combined utilization of hardware resources for concurrent checking and for off-line BIST is suggested in [6,36]. Moreover, BIST is ideally suited for hierarchical test structures (see Section 8.4).

In the face of continuously shrinking package size and pitch external probing becomes a virtually insoluble mechanical problem. At the same time system clock rates continue to rise and timing margins become narrower, therefore a test should also cover the dynamic properties of a circuit. Ideally, the test is performed at the nominal system clock rate (*at-speed testing* [93,94]), which also helps to keep test duration low. With a conventional external test approach, however, non-idealities in contacts and wiring deteriorate signal quality substantially, which prohibits at-speed testing. Moreover, the technology of the test equipment (clock rate, propagation delays) may be inferior to that of the CUT. These problems do not apply for BIST, since technology of CUT and tester are identical, probing is reduced to a minimum and timing of the interface signals is not critical. In some cases, however, the inclusion of the BIST logic exhibits a negative effect on system timing, resulting in the need for a minor reduction of the nominal system clock. It should further be noted, that at-speed testing alone does not adequately address delay faults. Accepted BIST methods for delay faults have not been proposed.

BIST significantly improves testability of a design: For the chip-internal tester all nodes in the circuit are directly accessible and test points can be freely chosen as suggested by the test strategy without having to connect them to external pins.

BIST facilitates a test-per-clock strategy, while conventional testing is often restricted to test-per-scan (see Section 7.3).

Another crucial cost factor is the degree of automation: VLSI design tools automatically generate the set of test vectors, but the adaptation or design process for the tester hardware and probing remains an expensive requirement for the conventional testing approach. As opposed to that there are suites of tools available that automatically add a complete and largely optimized BIST environment to a VHDL or Verilog-design [94].

Table 6 summarizes the above discussion on the benefits of BIST.

It is interesting to observe that BIST naturally unifies circuit design and test – two areas that have traditionally been largely disjoint. At the same time BIST moves test cost from equipment to design: The costly test equipment is traded for design overheads.

8.4. BIST and hierarchical testing

Perhaps the most significant benefit of BIST lies in its reusability on higher levels. A system level BIST may activate board level BIST on all system boards that, in turn, activate chip level BIST. The individual BIST results are combined and propagated through the hierarchy to finally form the system BIST result [9].

This approach has numerous advantages over the common software-implemented system test:

- *BIST localizes the testing problem*: There is no need to carry the test stimulus through many layers and then convey the result back again.
- As a design for testability method BIST is implemented already during circuit/board/system design by the expert who has the most detailed knowledge on the functionality of the design. Triggering the BIST from a higher level does not require any expertise on the lower level function.
- BIST naturally supports the hierarchical test approach that is often applied in conventional tests to reduce complexity.
- BIST allows for optimal diagnostic resolution.
- The extra cost for system level BIST can be kept at a minimum, since a reuse of the factory BIST logic on chip level is possible [95].

8.5. BIST and core design

An integrated circuit core is a pre-designed, pre-verified silicon circuit block, usually containing at least 5000 gates, that can be used in building a larger or more complex application on a semiconductor chip [96]. Popular core types include CPUs, RAMs and interface modules. The advantages of core designs become more and more obvious as the integration levels of ICs increase and the vision of a *system-on-a-chip* becomes reality: Cores facilitate design reuse and transfer of intellectual property. They allow the system integrator to concentrate on the product specific features, which shortens time to market significantly.

To protect his intellectual property, the core designer most often provides the system integrator with little or no knowledge of the internal structure of the core module. For this reason the core

Table 6
Comparison between BIST and conventional testing

	Conventional test	BIST
Probing	Difficult	Simple BIST interface
Selection of test points	Compromises	Free choice
Cost of test equipment	Very high	Chip area overhead
Technology	May be critical	Identical to CUT
Test speed	Problematic	"At-speed testing"
Delay faults	Testable	Not testable
Cost of test time	Often very high	Low
Applicability	Specific	Reusable
Automatic generation	Stimuli only	Complete

design must include testing facilities along with a clearly specified test interface. A boundary scan for the core module is one common way to provide test access. IEEE is currently developing a standardization framework for core testing (IEEE P1500). However, access to the chip is a necessary but not a sufficient condition to achieve good fault coverage, knowledge of chip-internal details is still required.

In this context BIST is an extremely attractive option: If a core is equipped with BIST, the interface is easy to specify while the intellectual property remains well protected. The core designer who has most detailed knowledge of the core function can implement the test and the BISTed core can be integrated as a black box into a hierarchical BIST concept below the chip level by the system integrator.

8.6. BIST and IDDQ-testing

The IDDQ-test has been pointed out as an effective method in Section 6.3. It is therefore not surprising that numerous attempts have been made to integrate IDDQ monitoring circuitry on a chip (e.g. "built-in current sensing" [49,97]). One crucial problem with built-in IDDQ testing for digital VLSI circuits, however, is the need for specific analog circuitry. For this reason the vast majority of BIST applications have focused on scan-techniques and/or functional testing for regular structures.

9. BIST for structured logic

Regular structures like PLAs, RAMs and ROMs play a special role in every test concept because most often the high regularity is paired with a simple, general functionality that allows (a) functional testing and (b) the application of generic test methods. It is therefore common to isolate these regular structures from the remaining random logic by extra *collar logic* and test them separately [9]. The respective generic test methods will be discussed in the following.

9.1. RAM testing

9.1.1. RAM fault model

Memories are commonly considered as the "most unreliable parts of the system" [98] and it is felt that, due to their extreme densities "RAMs require special test treatment" [9,13].

For the purpose of fault modeling a RAM can subdivided into the following functional units [12] (further discussions on RAM models can be found in [13,56,64,99,100]):

- memory cell array;
- interconnect lines;
- address decoder;
- read/write logic.

Since this is a functional model, no distinction between SRAMs and DRAMs must be made (unless data retention capability and refresh logic shall be explicitly tested). All functional faults listed in Table 4 can be assumed in the above units. However, a balance between comprehensiveness of the fault model on the one hand and test duration and test generation cost on the other hand must be found. For this reason embedded RAMs are typically tested for stuck-at faults only, while tests for stand-alone RAMs cover more complex fault models, because of their higher density [9]. Parametric faults are hardly ever tested for in a BIST environment.

In order to defeat overheads memory test is often reduced to a test of the memory cell array only. It is claimed that the array test also covers most faults in the other functional units. This claim has been partly invalidated by Sachdev [13] where it is shown that open defects in CMOS address decoders are not generally covered by array tests. However, specific March tests (see next section) have recently been proposed to cover these open faults as well.

9.1.2. Pattern testing

The functional test of a RAM is basically performed by read and write operations. Numerous algorithms have been proposed in the literature that are optimized for different fault models and different test structures. They differ in the algorithmic sequence for writing data to and reading data from the various address locations in the

RAM. Two test pattern generators are required: one for data and one for address. Since the functional mapping between write data and read data is one-to-one in the fault-free case, both test pattern generators could be employed upon read-back to automatically provide the reference data. However, to avoid the danger of a generator error (that might go undetected in this case), the usual compaction techniques are preferred for response checking. Since most test algorithms use very simple regular write data, compression techniques are also applicable [94].

Both random and deterministic testing are applied with a recent trend towards deterministic testing which facilitates an exact computation of fault coverage. Table 7 lists the most popular test algorithms and summarizes their properties as shown in [12], further discussions can be found in [99–104].

In the leftmost column of Table 7 all common test algorithms are listed by their name. For the sake of brevity functional explanations of the algorithms are not given here, these can be found in [12]. The fault coverage of the mechanisms with respect to different fault models (see Table 4) is shown in columns 2–6. The last two columns give an indication for the time required to perform the test: The column titled "1M" shows the time required to perform the respective test algorithm for a 1 Mbit memory assuming an access time of 100 ns. The column "order" indicates how this test duration is related to memory size. Linear dependence between memory size and test duration is denoted as n, if test time increases with the square of memory size, an order of n^2 is shown.

The class of *March-algorithms* is exceptionally advantageous for use with BIST: The complete address area is passed through sequentially for several times. This allows a simple counter to be employed as an address generator. Test data are also easy to generate: A sequence of "all 1s" or "all 0s" is written and read alternatively. Note that all March algorithms have an order of n and are significantly faster than traditional algorithms with comparable fault coverage.

A serial version of the March test is described in [102] that works similar to the circular BIST

Table 7
Comparison of memory test algorithms (Source: [12])

Algorithm	Fault coverage					Test time	
	Stuck-at	Address	Transition	Coupling	Others	Order	1M
Zero-one	Locate all					n	0.42 s
Checkerboard	Locate all				Refresh	n	0.52 s
Walking 1/0	Locate all	Locate all	Locate all	Locate all	Sense amplif. recovery	n^2	2.5 d
GALPAT	Locate all	Locate all	Locate all	Locate all	Write recovery	n^2	5.1 d
GALROW	Locate all	Locate some	Locate all	Locate all	Write recovery	$n\sqrt{n}$	7.2 m
GALCOL	Locate all	Locate some	Locate all	Locate all	Write recovery	$n\sqrt{n}$	7.2 m
Sliding diagonal	Locate all	Locate some	Locate all			$n\sqrt{n}$	3.6 m
Butterfly	Locate all	Locate some	Locate all			$n\log_2(n)$	10 s
MATS	Detect all	Detect some				n	0.42 s
MATS+	Detect all	Detect all				n	0.52 s
Marching 1/0	Detect all	Detect all	Detect all			n	1.5 s
MATS++	Detect all	Detect all	Detect all			n	0.63 s
March X	Detect all	Detect all	Detect all	Detect all		n	0.63 s
March C−	Detect all	Detect all	Detect all	Detect all		n	1.0 s
March A	Detect all	Detect all	Detect all	Detect all		n	1.6 s
March Y	Detect all	Detect all	Detect all	Detect all		n	0.85 s
March B	Detect all	Detect all	Detect all	Detect all		n	1.8 s
MOVI	Detect all	Detect all	Detect all	Detect all	Read access time	$n\log_2(n)$	25 s
3-coupling	Detect all	Detect all	Detect all	Detect all	3-coupling faults	$n\log_2(n)$	54 s
Paragon	Detect all	Detect all	Detect all	Detect all	Operational faults	n^2	20 d

described in Section 7.3.2: Upon read-back of the previous memory contents the LSB is fed into a signature register and the remaining bits of the data word are shifted to the right by one position, which leaves room to shift in a new bit from the test pattern generator to the MSB position. The data word thus generated is used as a new input pattern in the next step. In this way the bit sequence generated by the test pattern generator is continuously shifted along the memory cells like in a shift register, until it is finally fed into the signature register where the response analysis is performed. This method of using the CUT for test pattern generation makes the implementation extremely compact, while the simplicity of the data transformation (a shift operation) facilitates analytical treatment.

All standard memory test procedures rely on an initialization of the whole memory or at least part of it before the actual test is started. The destruction of the memory contents implied by this initialization, however, is inadmissible in the case of periodic on-line testing. For this purpose *transparent testing* must be applied: Any of the standard algorithms can be modified such that an initialization may be dropped and the memory contents are preserved by the test [105,106].

The performance of a memory test algorithm is an important issue in the context of certification of a fault-tolerant system for safety-critical applications. In [35,39] a subset of the above mechanisms is classified with respect to coverage.

9.1.3. IDDQ test

Due to the highly regular structure of CMOS RAMs the IDDQ test as described in Section 6.3 is a particularly attractive option for their test; however, the analog circuitry involved makes a BIST implementation difficult.

9.1.4. Specific DPM aspects

A Dual-Ported RAM can be tested by performing standard memory tests for both sides. To additionally test the dual-port functionality, these tests must be extended by additional reads from the respective other side.

9.2. ROM testing

9.2.1. ROM fault model

For the purpose of testing ROMs are commonly modeled either on the logical level or on the functional level:

- Logical Model: A ROM is a combinational circuit that transforms every address word applied at the input into a data word that can be read at the output. This mapping can be described by a function table.
- Functional Model: A ROM is a non-volatile memory. Reading from one given address must always produce the same data. These data are known a priori. This makes the ROM test much easier than a RAM test and allows checking data integrity in addition to the proper functionality of the device.

It must be noted here, that the above models apply for a *programmed* ROM in its application. If the ROM is to be tested generally as an electronic component (factory test), the test must also cover the write logic of the ROM, which makes ROM testing an equally complex problem as RAM testing, aggravated by the fact that access time for ROMs is traditionally higher than for RAMs [100].

9.2.2. Structural ROM testing

Like any other combinational logic an embedded ROM can be tested structurally by test patterns via the scan path. Due to its simplicity this method is very usual. It is, however, not capable of detecting sequential faults (i.e. faults making combinational logic behave like a sequential one) that might be induced by open defects.

9.2.3. Signature computation

An exhaustive functional test of the ROM [86] can be performed quite easily by reading the ROM contents and compacting them to a signature by means of a multiple input shift register (MISR, see Section 7.4). Multiple bit faults originating from defects in decoder or output register may escape detection if aliasing occurs [107]. Several suggestions have been made to reduce the aliasing probability in this case:

- a few deterministic test-vectors can be added to detect typical multiple bit fault scenarios [9], or
- an additional bit can be appended to each data word that not only reduces aliasing probability but also allows concurrent checking of the ROM contents [36].

It is interesting to note that structural testing and signature computation are very similar in the sense that the test patterns applied during the structural test are nothing else than the addresses, and the response is compacted by a signature calculation in both cases. With neither method a guarantee for detecting sequential faults can be given, although reversing the read order in the functional test has the potential to cover a high proportion of them.

9.3. PLA testing

Basically PLAs can be modeled and tested just like ROMs [5]: They perform a combinational mapping between input patterns and output patterns. Therefore, exhaustive pattern testing is the best solution. This is, however, economic only for PLAs with less than 25 inputs. For larger PLAs currently no general BIST solutions exist with a reasonable balance between cost and coverage. If partitioning of the logic is possible, the pseudo-exhaustive technique can be applied as described in Section 7.2.3.

It has been found that cross point faults are most prevalent in PLA structures, therefore special methods have been devised to facilitate detection of this type of fault [9,108,109]. Research has shown that methods detecting cross point faults also detect most other classical faults (stuck-at faults and bridging faults are of special interest in PLAs).

10. Conclusion

The important role of testing in today's VLSI designs has been pointed out in this survey, and numerous indications have been given that this role will become even more dominant in the future.

At the same time conventional test technology is reaching its limits, particularly with respect to probing. BIST advances the state of the art in testing by integrating the required test logic into the circuit under test. This keeps the external interface simple, while facilitating the application of highly efficient test strategies. Many of the highly advanced concepts of the traditional test can directly be applied for BIST. As an additional advantage the integration of the test logic on the chip allows its reuse in different phases of product testing. In this way BIST provides not only an economic solution for start-up and on-line testing but also an ideal foundation for hierarchic system test.

References

[1] R. Bennetts, Design of Testable Logic Circuits, Addison-Wesley, Reading, MA, 1984.
[2] R. Williams, IBM perspectives on the electrical design automation industry, in: Keywords to IEEE Design Automation Conference, 1986.
[3] B. Könemann, Creature from the Deep Submicron Lagoon, in: Keywords to the 10th ITG Workshop on Testmethoden und Zuverlässigkeit von Schaltungen, Herrenberg, Germany, March 1998.
[4] Semiconductor Industry Association, The National Technology Roadmap for Semiconductors, 1997 edition, http://www.sematech.org.
[5] P. Bardell, W. McAnney, J. Savir, Built-in Test for VLSI, Pseudorandom Techniques, Wiley, New York, 1987.
[6] X. Sun, M. Serra, Merging concurrent checking and off-line BIST, in: Proceedings of International Test Conference, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 958–967.
[7] E. Amerasekera, F. Najm, Failure Mechanisms in Semiconductor Devices, second ed., Wiley, Chichester, 1997.
[8] Siemens, Fourth Generation 16M-Bit DRAMs – Accelerated Soft Error Sensitivity, Information Note 5.96.
[9] V. Agrawal, C. Kime, K. Saluja, A tutorial on built-in self-test, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, March 1993, pp. 73–80 and June 1993, pp. 69–77.
[10] P. Maxwell, R. Aitken, V. Johansen, I. Chiang, The effectiveness of IDDQ, functional and scan tests: how many fault coverages do we need? in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 168–177.
[11] C. Chen, S. Gupta, BIST test pattern generators for two-pattern testing – theory and design algorithms, IEEE Transactions on Computers 45 (3) (1996) 257–269.
[12] A. van de Goor, Testing Semiconductor Memories, Theory and Practice, Wiley, Chichester, 1991.

[13] M. Sachdev, Open defects in CMOS RAM address decoders, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, April 1997, pp. 26–33.
[14] B. Könemann et al., Delay test: the next frontier for LSSD test systems, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 578–587.
[15] E. McCluskey, Built-in self-test techniques, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, April 1985, pp. 21–36.
[16] R. Fritzemeier, H. Nagle, C. Hawkins, Fundamentals of testability – a tutorial, IEEE Transactions on Industrial Electronics 36 (2) (1989) 117–128.
[17] J. Grason, TMEAS, A Testability Measurement Program, in: Proceedings of 16th Design Automation Conference, June 1979, San Diego, CA, pp. 156–161.
[18] L. Goldstein, E. Thigpen, SCOAP: Sandia Controllability/Observability Analysis Program, in: Proceedings of 17th ACM/IEEE Design Automation Conference, June 1980, Minneapolis, MN, pp. 190–196.
[19] I. Ratiu, A. Sangiovanni-Vincentelli, D. Pederson, VICTOR: A Fast VLSI Testability Analysis Program, in: Proceedings of International Test Conference 1982, November 1982, Philadelphia, PA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 397–401.
[20] T. Williams, Design for testability, in: T. Williams (Ed.), Advances in CAD for VLSI vol. 5: VLSI Testing, Elsevier, Amsterdam, 1986.
[21] Logic Vision, ICBIST User Guide, 1995.
[22] H. Nagle, S. Roy, C. Hawkins, M. McNamer, R. Fritzemeier, Design for testability and built-in self test: a review, IEEE Transactions on Industrial Electronics 36 (2) (1989) 129–140.
[23] C. Hawkins, H. Nagle, R. Fritzemeier, J. Guth, The VLSI test problem – a tutorial, IEEE Transactions on Industrial Electronics 36 (2) (1989) 111–116.
[24] T. Williams, K. Parker, Design for testability – a survey, in: D. Siewiorek, S. Swarz (Eds.), Reliable Computer Systems: Design and Evaluation, second ed., Digital Press, Belford, MA, 1992, pp. 803–830.
[25] S. Kajihara, H. Shiba, K. Kinoshita, Removal of redundancy in logic circuits under classification of undetectable faults, in: Proceedings of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), 8–10 July 1992, Boston, MA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 263–270.
[26] E. Eichelberger, et al., Structured Logic Testing, Prentice-Hall, Englewood Cliffs, NJ, 1991.
[27] B. Seiss, P. Trouborst, M. Schult, Test point insertion for scan-Based BIST, in: Proceedings of International Test Conference 1991, IEEE Computer Soc. Press, Silver Spring, MD, pp. 253–262.
[28] L. Youngs, S. Paramanadam, Mapping and repairing embedded-memory defects, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, January 1997, pp. 18–24.
[29] Bath Scientific, LinearProbe – Production Ready Fixtureless Testing. http://www.bathsl.com.
[30] Probetest Systems Ltd., The APS8000 Series, http://www.probetest.com.
[31] IEEE Standard 1149.5: Standard Module Test and Maintenance Bus (Draft), IEEE 1994.
[32] D. Landis, C. Hudson, P. McHugh, Applications of the IEEE P1149.5 module test and maintenance bus, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 984–992.
[33] R. Tulloss, Leaving the wires to last – a functional evaluation of the IEEE standard test and maintenance bus, in: Proceedings of International Test Conference 1995, Los Alamitos, CA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 797–806.
[34] D. Siewiorek, S. Swarz, Reliable Computer Systems: Design and Evaluation, second ed., Digital Press, Belford, MA, 1992.
[35] DIN/VDE/ÖVE 0801/01.90: Grundsätze für Rechner in Sicherheitsaufgaben.
[36] S. Gupta, D. Pradhan, Can concurrent checkers help BIST? in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 140–150.
[37] S. Vanstone, P. Oorschot, An Introduction to Error Correcting Codes with Applications, Kluwer Academic Publishers, Dordrecht, 1992.
[38] M. Rela, H. Madeira, J. Silva, Experimental evaluation of the fail-silent behavior in programs with consistency checks, in: Proceedings of 26th International Symposium on Fault-Tolerant Computing (FTCS-26), June 1996, Sendai, Japan, IEEE Computer Soc. Press, Silver Spring, MD, pp. 394–403.
[39] H. Hoelscher, J. Rader, Microcomputers in Safety Technique, TUEV, Bayern.
[40] Reliability Prediction of Electronic Equipment, Military Handbook MIL-HDBK-217, United States Department of Defense.
[41] K. Roy et al., Relative effectiveness of tests, in: Roundtable in IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, January 1998, pp. 83–90.
[42] H. Nagle, R. Fritzemeier, J. van Well, M. McNamer, Microprocessor testability, IEEE Transactions on Industrial Electronics 36 (2) (1989) 151–163.
[43] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines – a survey, in: Proceedings of the IEEE 84 (8) (1996) 1090–1123.
[44] J. Soden, C. Hawkins, IDDQ testing: issues present and future, in: IEEE Design & Test of Computers, Winter/1996, pp. 61–65.
[45] K. Sawada, S. Kayano, An evaluation of IDDQ versus conventional testing for CMOS Sea-of-Gate IC's, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 158–167.

[46] R. Perry, IDDQ testing in CMOS digital ASIC's – putting it all together, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 151–157.

[47] C. Hawkins, J. Soden, R. Fritzemeier, L. Horning, Quiescent power supply current measurement for CMOS IC defect detection, IEEE Transactions on Industrial Electronics 36 (2) (1989) 211–218.

[48] R. Gulati, W. Mao, D. Goel, Detection of 'undetectable' faults using IDDQ testing, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 770–777.

[49] F. Vargas, M. Nicolaidis, SEU-tolerant SRAM design based on current monitoring, in: Proceedings of 24th International Symposium on Fault-Tolerant Computing (FTCS-24), 15–17 June 1994, Austin, Texas, IEEE Computer Soc. Press, Silver Spring, MD, pp. 106–115.

[50] J. Abraham, Fault modeling in VLSI, in: T. Williams (Ed.), Advances in CAD for VLSI, vol. 5: VLSI Testing, Elsevier, Amsterdam, 1986.

[51] P. Bottorff, Test generation and fault simulation, in: T. Williams (Ed.), Advances in CAD for VLSI, vol. 5: VLSI Testing, Elsevier, Amsterdam, 1986.

[52] M. Sheu, C. Lee, Simplifying sequential circuit test generation, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, Fall 1994, pp. 29–38.

[53] I. Pomeranz, S. Reddy, LOCSTEP: A logic simulation based test generation procedure, in: Proceedings of 25th International Symposium on Fault-Tolerant Computing (FTCS-25), 27–30 June 1995, Pasadena, CA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 110–119.

[54] T. Fujino, H. Fujiwara, An efficient test generation algorithm based on search state dominance, in: Proceedings of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), 8–10 July 1992, Boston, MA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 246–253.

[55] J. Roth, W. Bouricius, P. Schneider, Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits, IEEE Transactions on Electronic Computers EC-16 (October) (1967) 547–580.

[56] H. Wunderlich, Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test, Springer, Berlin, 1991.

[57] H. Fujiwara, T. Shimono, On the acceleration of test generation algorithms, in: Proceedings of 13th International Symposium on Fault-Tolerant Computing (FTCS-13), 28–30 June 1983, Milan, Italy, IEEE Computer Soc. Press, Silver Spring, MD, pp. 98–105.

[58] G. Robinson, HITEST – intelligent test generation, Proc. International Test Conference 1983, September 1983, Philadelphia, PA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 311–323.

[59] P. Goel, An implicit enumeration algorithm to generate tests for combinational logic circuits, IEEE Transactions on Computers C-30 (3) (1981) 215–222.

[60] J. Gaisler, Concurrent error-detection and modular fault-tolerance in a 32-bit processing core for embedded space flight applications, in: Proceedings of 24th International Symposium on Fault-Tolerant Computing (FTCS-24), 15–17 June 1994, Austin, Texas, IEEE Computer Soc. Press, Silver Spring, MD, pp. 128–130.

[61] I. Pomeranz, S. Reddy, A divide-and-conquer approach to test generation for large synchronous sequential circuits, in: Proceedings of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), 8–10 July 1992, Boston, MA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 230–237.

[62] G. Al-Hayek, Y. Le Traon, C. Robach, Impact of system partitioning on test cost, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, January 1997, pp. 64–74.

[63] B. Murray, J. Hayes, Testing ICs: getting to the core of the problem, IEEE Computer 29 (11) (1996) 32–38.

[64] A. Fuentes, R. David, B. Courtois, Random testing versus deterministic testing of RAMs, in: Proceedings of 16th International Symposium on Fault-Tolerant Computing (FTCS-16), 1–4 July 1986, Vienna, Austria, IEEE Computer Soc. Press, Silver Spring, MD, pp. 266–271.

[65] S. Mukund, E. McCluskey, T. Rao, An apparatus for pseudo-deterministic testing, CRC Report No. 94-12, Center for Reliable Computing, Stanford University, October 1994.

[66] C. Dufaza, G. Cambon, LFSR-based deterministic and pseudo-random test pattern generator structures, in: Proceedings of European Test Conference 1991, IEEE Computer Soc. Press, Silver Spring, MD, pp. 27–34.

[67] J. Savir, W. McAnney, A multiple seed linear feedback shift register, IEEE Transactions on Computers 41 (2) (1992) 250–252.

[68] S. Hellebrand, S. Tarnick, J. Rajski, Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers, Proc. International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 120–129.

[69] P. Hortensius, H. Card, R. McLeod, W. Pries, Parallel random number generation for VLSI using cellular automata, IEEE Transactions on Computers (6) (1989) 769–774.

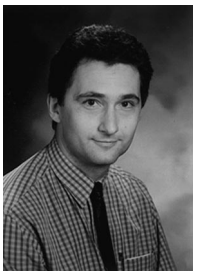
[70] J. van Sas, F. Catthoor, H. De Man, Optimized BIST strategies for programmable data paths based on cellular automata, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 110–119.

[71] B. Koenemann, J. Mucha, G. Zwiehoff, Built-in logic block observation techniques, in: Proceedings of International Test Conference 1979, Cherry Hill, NJ, pp. 37–41.

[72] L. Nachmann, K. Saluja, S. Upadhyaya, R. Reuse, Random pattern testing for sequential circuits revisited, in: Proceedings of 26th International Symposium on Fault-Tolerant Computing (FTCS-26), June 1996, Sendai, Japan, IEEE Computer Soc. Press, Silver Spring, MD, pp. 44–52.

[73] F. Corno, P. Prinetto, M. Reorda, Circular self-test path for FSMs, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, Winter 1996, pp. 50–60.
[74] A. Krasniewski, S. Pilarski, High quality testing of embedded RAMs using circular self-test path, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 652–661.
[75] Synopsis, Scan Synthesis/User Guide, August 1997.
[76] E. Eichelberger, T. Williams, A logic design structure for LSI testability, in: Proceedings of 14th Design Automation Conference, June 1977, pp. 462–468.
[77] H. Ando, Testing VLSI with random access scan, in: Proceedings of Compcon 80, 1980, pp. 50–52.
[78] J. Steward, Application of SCAN/SET for error detection and diagnostics, in: Proceedings of International Test Conference 1978, IEEE Computer Soc. Press, Silver Spring, MD.
[79] K. Cheng, Single-clock partial scan, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, Summer 1995, pp. 24–31.
[80] J. Rajski, J. Tyszer, Accumulator-based compaction of test responses, IEEE Transactions on Computers 42 (6) (1993) 643–649.
[81] J. Savir, Syndrome-testable design of combinational circuits, IEEE Transactions on Computers 29 (3) (1980) 442–451.
[82] A. Susskind, Testing by verifying Walsh coefficients, IEEE Transactions on Computers C-32 (2) (1983) 198–201.
[83] S. Hassan, E. McCluskey, Increased fault coverage through multiple signatures, in: Proceedings of 14th International Symposium on Fault-Tolerant Computing (FTCS-14), 20–22 June 1984, Kissimmee, Florida, IEEE Computer Soc. Press, Silver Spring, MD, pp. 354–359.
[84] N. Saxena, E. McCluskey, Parallel signature analysis design with bounds on aliasing, IEEE Transactions on Computers 46 (4) (1997) 425–438.
[85] Y. Zorian, V. Agarwal, Optimizing error masking in BIST by output data modification, Journal on Electronic Testing: Theory and Applications 1 (February) (1990) 59–72.
[86] Y. Zorian, A. Ivanov, An effective BIST scheme for ROM's, IEEE Transactions on Computers 41 (5) (1992) 646–653.
[87] K. Aiyama, K.K. Saluja, A method of reducing aliasing in built-in self-test environment, IEEE Transactions on Computer-Aided Design CAD-10 (4) (1991) 548–553.
[88] IEEE Standard 1149.1: A Standard Boundary Scan Architecture.
[89] C. Maunder, R. Tulloss, The Test Access Port and Boundary Scan Architecture, IEEE Computer Society Press, Los Alamitos, CA, 1992.
[90] K. Parker, The Boundary Scan Handbook, Kluwer Academic Publishers, Norwell, MA, 1992.
[91] M. Levitt, Designing Ultra Sparc for testability, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, January 1997, pp. 10–17.
[92] S. Barbagallo, D. Medina, F. Corno, P. Prinetto. M. Reorda, Integrating online and offline testing of a switching memory, IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, January 1998, pp. 63–70.
[93] B. Nadeau-Dostie, D. Burek, A. Hassan, ScanBist: a multifrequency scan-based BIST method, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, Spring 1994, pp. 7–17.
[94] B. Könemann, B. Bennetts, N. Jarwala, B. Nadeau-Dostie, Built-in self-test: assuring system integrity, IEEE Computer 29 (11) (1996) 39–45.
[95] M. Lubaszewski, B. Courtois, On the design of self-checking boundary scannable boards, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 372–381.
[96] R. Gupta, Y. Zorian, Introducing core-based system design, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, Winter 1997, pp. 15–25.
[97] J. Lo, J. Daly, M. Nicolaidis, Design of static CMOS self-checking circuits using built-in current sensing, in: Proceedings of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), 8–10 July 1992, Boston, MA, IEEE Computer Soc. Press, Silver Spring, MD, pp. 104–111.
[98] F. Corno et al., On-line testing of an off-the-shelf microprocessor board for safety-critical applications, in: Proceedings of Second European Dependable Computing Conference (EDCC-2), 2–4 October 1996, Taormina, Italy, Springer, pp. 190–201.
[99] P. Camurati, P. Prinetto, M. Reorda, Industrial BIST of embedded RAMs, in: IEEE Design & Test of Computers, IEEE Computer Soc. Press, Silver Spring, MD, Fall 1995, pp. 86–95.
[100] A. Tuszynski, Memory testing, in: T. Williams (Ed.), Advances in CAD for VLSI, vol. 5: VLSI Testing, Elsevier, Amsterdam, 1986.
[101] A. van de Goor, Using March tests to test SRAMs, IEEE Design & Test of Computers, March 1993, pp. 8–14.
[102] B. Nadeau-Dostie, A. Silburt, V. Agarwal, Serial interfacing for embedded memory testing, in: IEEE Design & Test of Computers, April 1990, pp. 52–63.
[103] P. Mazumder, J. Patel, An efficient built-in self testing for random access memory, IEEE Transactions on Industrial Electronics 36 (2) (1989) 246–253.
[104] S. Jain, C. Stroud, Built-in self testing of embedded memories, in: IEEE Design & Test, October 1986, pp. 27–37.
[105] M. Nicolaidis, Theory of transparent BIST for RAMs, IEEE Transactions on Computers 45 (10) (1996).
[106] M. Nicolaidis, Transparent BIST for RAMs, in: Proceedings of International Test Conference 1992, 20–24 September 1992, Baltimore, MD, IEEE Computer Soc. Press, Silver Spring, MD, pp. 598–607.

[107] K. Iwasaki, S. Nakamura, Aliasing error for a mask ROM built-in self-test, IEEE Transactions on Computers 45 (3) (1996) 270–277.
[108] V. Agarwal, Easily testable PLA design, in: T. Williams (Ed.), Advances in CAD for VLSI, vol. 5: VLSI Testing, Elsevier, Amsterdam, 1986.
[109] R. Treuer, H. Fujiwara, V. Agarwal, A low overhead, high coverage built-in self-test PLA design, in: Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTCS-15), June 1985, Ann Arbor, Michigan, IEEE Computer Soc. Press, Silver Spring, MD, pp. 112–117.



Andreas Steininger is Associate Professor at the Vienna University of Technology where he is currently leading a VLSI design group. Within the last decade he has been involved in many projects concerned with real-time communication networks, the design of fault-tolerant hardware architectures and their evaluation by means of fault-injection. His research interests further include on-line testing and built-in self-test. Steininger received his MS degree in Electrical Engineering and the Ph.D. degree in Computer Science, both from the Vienna University of Technology.