
Online BIST for Embedded Systems

HUSSAIN AL-ASAAD

University of California, Davis

BRIAN T. MURRAY

General Motors

JOHN P. HAYES

University of Michigan,

Ann Arbor

EMBEDDED SYSTEMS are computers incorporated in consumer products or other devices to perform application-specific functions. The product user is usually not even aware of the existence of these systems. From toys to medical devices, from ovens to automobiles, the range of products incorporating microprocessor-based, software-controlled systems has expanded rapidly since the introduction of the microprocessor in 1971. The lure of embedded systems is clear: They promise previously impossible functions that enhance the performance of people or machines. As these systems gain sophistication, manufacturers are using them in increasingly critical applications—products that can result in injury, economic loss, or unacceptable inconvenience when they do not perform as required.

Embedded systems can contain a variety of computing devices, such as microcontrollers, application-specific integrated circuits, and digital signal processors. A key requirement is that these computing devices continuously respond to external events in real time. Makers of embedded systems take many measures to ensure safety and reliability throughout the lifetime of products incorporating the systems. Here, we consider techniques for identifying faults during normal operation of the product—that is, on-line-testing techniques. We evaluate them on the basis of error coverage, error latency, space redundancy, and time redundancy.

Embedded-system test issues

Cost constraints in consumer products typically translate into stringent constraints on

product components. Thus, embedded systems are particularly cost sensitive. In many applications, low production and maintenance costs are as important as performance.

Moreover, as people become dependent on computer-based systems, their expectations of these systems' availability increase dramatically. Nevertheless, most people still expect significant downtime with computer systems—perhaps a few hours per month. People are much less patient with computer downtime in other consumer products, since the items in question did not demonstrate this type of failure before embedded systems were added. Thus, complex consumer products with high availability requirements must be quickly and easily repaired. For this reason, automobile manufacturers, among others, are increasingly providing online detection and diagnosis, capabilities previously found only in very complex and expensive applications such as aerospace systems. Using embedded systems to incorporate functions previously considered exotic in low-cost, everyday products is a growing trend.

Since embedded systems are frequently components of mobile products, they are exposed to vibration and other environmental stresses that can cause them to fail. Embedded systems in automotive applications are exposed to extremely harsh environments, even beyond those experienced by most portable devices.

Table 1 (next page) summarizes the attributes of microprocessor applications that affect ordinary consumers. These applications are proliferating rapidly, and their more

Embedded systems must meet increasingly high expectations of safety and high reliability. The authors survey on-line-testing techniques for identifying faults that can lead to system failure. They focus on on-line built-in self-test and its role in a comprehensive testing approach.

Table 1. Microprocessor applications affecting ordinary consumers (excludes industrial applications such as process controllers).

Application	Embedded?	Cost sensitive?	Portable?	Harsh environment?	Highly available?	Critical?
Desktop PC						
Laptop PC			✓			
TV set-top box	✓	✓			✓	
Home security	✓				✓	
Mobile phone	✓	✓	✓		✓	
Home appliance	✓	✓			✓	
Automotive: engine/chassis/safety	✓	✓	✓	✓	✓	✓
Automotive: comfort/convenience	✓	✓	✓	✓	✓	
Medical implant	✓		✓	✓	✓	✓
Smart card	✓		✓		✓	

stringent safety and reliability requirements pose a significant challenge for designers. Critical applications and applications with high availability requirements are the main candidates for online testing.

Embedded systems consist of hardware and software, each usually considered separately in the design process, despite progress in the field of hardware-software codesign. A strong synergy exists between hardware and software failure mechanisms and diagnosis, as in other aspects of system performance. System failures often involve defects in both hardware and software. Software does not “break” in the common sense of the term. However, it can perform inappropriately due to faults in the underlying hardware or specification or design flaws in either hardware or software. At the same time, one can exploit the software to test for and respond to the presence of faults in the underlying hardware.

Online software testing aims at detecting design faults (bugs) that avoid detection before the embedded system is incorporated and used in a product. Even with extensive testing and formal verification of the system, some bugs escape detection. Residual bugs in well-tested software typically behave as intermittent faults, becoming apparent only in rare system states. Online software testing relies on two basic methods: acceptance testing and diversity.¹ Acceptance testing checks for the presence or absence of well-defined events or conditions, usually expressed as true-or-false conditions (predicates), related to the correctness or safety of preceding computations. Diversity techniques compare replicated computations, either with minor variations in data (data diversity) or with proce-

dures written by separate, unrelated design teams (design diversity).

This article focuses on digital hardware testing, including techniques by which hardware tests itself, built-in self-test (BIST). Nevertheless, we must consider the role of software in detecting, diagnosing, and handling hardware faults. If we can use software to test hardware, why should we add hardware to test hardware? There are two possible answers. First, it may be cheaper or more practical to use hardware for some tasks and software for others. In an embedded system, programs are stored online in hardware-implemented

memories such as ROMs (for this reason, embedded software is sometimes called firmware). This program storage space is a finite resource whose cost is measured in exactly the same way as other hardware. A function such as a test is “soft” only in the sense that it can easily be modified or omitted in the final implementation.

The second answer involves the time that elapses between a fault’s occurrence and a problem arising from that fault. For instance, a fault may induce an erroneous system state that can ultimately lead to an accident. If the elapsed time between the fault’s occurrence and the corresponding accident is short, the fault must be detected immediately. Acceptance tests can detect many faults and errors in both software and hardware. However, their exact fault coverage is hard to measure, and even when coverage is complete, acceptance tests may take a long time to detect some faults. BIST typically targets relatively few hardware faults, but it detects them quickly.

These two issues, cost and latency, are the main parameters in deciding whether to use hardware or software for testing and which hardware or software technique to use. This decision requires system-level analysis. We do not consider software methods here. Rather, we emphasize the appropriate use of widely implemented BIST methods for online hardware testing. These methods are components in the hardware-software trade-off.

Online testing

Faults are physical or logical defects in the design or implementation of a digital device. Under certain conditions,

they lead to errors—that is, incorrect system states. Errors induce failures, deviations from appropriate system behavior. If the failure can lead to an accident, it is a hazard. Faults can be classified into three groups: design, fabrication, and operational. Design faults are made by human designers or CAD software (simulators, translators, or layout generators) during the design process. Fabrication defects result from an imperfect manufacturing process. For example, shorts and opens are common manufacturing defects in VLSI circuits. Operational faults result from wear or environmental disturbances during normal system operation. Such disturbances include electromagnetic interference, operator mistakes, and extremes of temperature and vibration. Some design defects and manufacturing faults escape detection and combine with wear and environmental disturbances to cause problems in the field.

Operational faults are usually classified by their duration:

- *Permanent faults* remain in existence indefinitely if no corrective action is taken. Many are residual design or manufacturing faults. The rest usually occur during changes in system operation such as system start-up or shutdown or as a result of a catastrophic environmental disturbance such as a collision.
- *Intermittent faults* appear, disappear, and reappear repeatedly. They are difficult to predict, but their effects are highly correlated. When intermittent faults are present, the system works well most of the time but fails under atypical environmental conditions.
- *Transient faults* appear and disappear quickly and are not correlated with each other. They are most commonly induced by random environmental disturbances.

One generally uses online testing to detect operational faults in computers that support critical or high-availability applications. The goal of online testing is to detect fault effects, or errors, and take appropriate corrective action. For example, in some critical applications, the system shuts down after an error is detected. In other applications, error detection triggers a reconfiguration mechanism that allows the system to continue operating, perhaps with some performance degradation. Online testing can take the form of external or internal monitoring, using either hardware or software. Internal monitoring, also called self-testing, takes place on the same substrate as the circuit under test (CUT). Today, this usually means inside a single IC—a system on a chip.

There are four primary parameters to consider in designing an online-testing scheme:

- *error coverage*—the fraction of modeled errors detected, usually expressed as a percentage. Critical and highly available systems require very good error coverage

to minimize the probability of system failure.

- *error latency*—the difference between the first time an error becomes active and the first time it is detected. Error latency depends on the time taken to perform a test and how often tests are executed. A related parameter is fault latency, the difference between the onset of the fault and its detection. Clearly, fault latency is greater than or equal to error latency, so when error latency is difficult to determine, test designers often consider fault latency instead.
- *space redundancy*—the extra hardware or firmware needed for online testing.
- *time redundancy*—the extra time needed for online testing.

The ideal online-testing scheme would have 100% error coverage, error latency of 1 clock cycle, no space redundancy, and no time redundancy. It would require no redesign of the CUT and impose no functional or structural restrictions on it. Most BIST methods meet some of these constraints without addressing others. Considering all four parameters in the design of an online-testing scheme may create conflicting goals. High coverage requires high error latency, space redundancy, and/or time redundancy. Schemes with immediate detection (error latency equaling 1) minimize time redundancy but require more hardware. On the other hand, schemes with delayed detection (error latency greater than 1) reduce time and space redundancy at the expense of increased error latency. Several proposed delayed-detection techniques assume equiprobable input combinations and try to establish a probabilistic bound on error latency.² As a result, certain faults remain undetected for a long time because tests for them rarely appear at the CUT's inputs.

To cover all the operational fault types described earlier, test engineers use two different modes of online testing: concurrent and nonconcurrent. Concurrent testing takes place during normal system operation, and nonconcurrent testing takes place while normal operation is temporarily suspended. One must often overlap these test modes to provide a comprehensive online-testing strategy at acceptable cost. Figure 1 (next page) depicts a taxonomy of online-testing techniques for embedded systems.

Nonconcurrent testing. This form of testing is either event-triggered (sporadic) or time-triggered (periodic) and is characterized by low space and time redundancy. Event-triggered testing is initiated by key events or state changes such as start-up or shutdown, and its goal is to detect permanent faults. Detecting and repairing permanent faults as soon as possible is usually advisable. Event-triggered tests resemble manufacturing tests. Any such test can be applied online, as long as the required testing resources are avail-

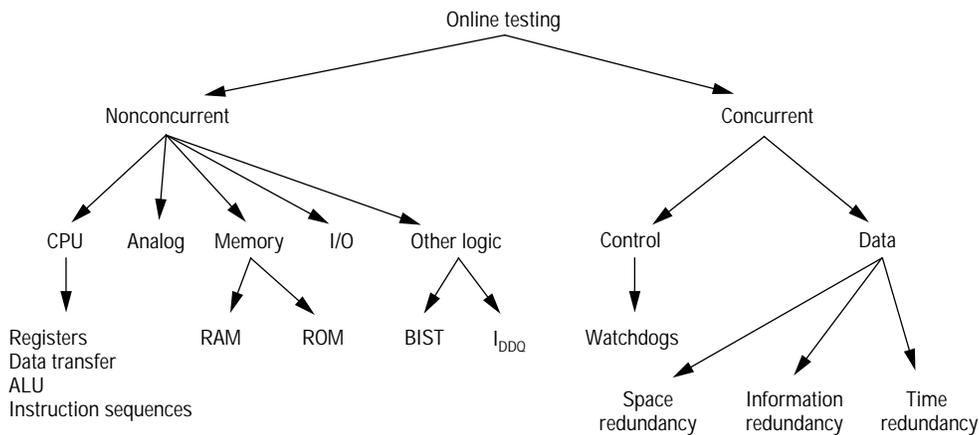


Figure 1. Taxonomy of online-testing methods for embedded systems.

able. Typically, the hardware is partitioned into components, each exercised by specific tests. RAMs, for instance, are tested with manufacturing tests such as March tests.³

Time-triggered testing occurs at predetermined times in the operation of the system. It detects permanent faults, often using the same types of tests applied by event-triggered testing. The periodic approach is especially useful in systems that run for extended periods during which no significant events occur to trigger testing. Periodic testing is also essential for detecting intermittent faults. Such faults typically behave as permanent faults for short periods. Since they usually represent conditions that must be corrected, diagnostic resolution is important. Periodic testing can identify latent design or manufacturing flaws that appear only under certain environmental conditions. Time-triggered tests are frequently partitioned and interleaved so that only part of the test is applied during each test period.

Concurrent testing. Nonconcurrent testing cannot detect transient or intermittent faults whose effects disappear quickly. Concurrent testing, on the other hand, continuously checks for errors due to such faults. However, concurrent testing is not particularly useful for diagnosing the source of errors, so test designers often combine it with diagnostic software. They may also combine concurrent and nonconcurrent testing to detect or diagnose complex faults of all types.

A common method of providing hardware support for concurrent testing, especially for detecting control errors, is a watchdog timer.⁴ This is a counter that the system resets repeatedly to indicate that the system is functioning properly. The watchdog concept assumes that the system is fault-free—or at least alive—if it can reset the timer at appropriate intervals. The ability to perform this simple task implies that control flow is correctly traversing timer reset points. One can monitor system sequencing very precisely by guarding the watch-

dog-reset operations with software-based acceptance tests that check signatures computed while control flow traverses various checkpoints. To implement this last approach in hardware, one can construct more complex hardware watchdogs.

A key element of concurrent testing for data errors is redundancy. For example, the duplication-with-comparison (DWC) technique⁵ detects any single error at the expense of 100% space redundancy. This technique re-

quires two copies of the CUT, which operate in tandem with identical inputs. Any discrepancy in their outputs indicates an error. In many applications, DWC's high hardware overhead is unacceptable. Moreover, it is difficult to prevent minor timing variations between duplicated modules from invalidating comparisons.

A possible lower-cost alternative is time redundancy. A technique called double execution, or retry, executes critical operations more than once at diverse time points and compares their results. Transient faults are likely to affect only one instance of the operation and thus can be detected. Another technique, recomputing with shifted operands (RESO),⁵ achieves almost the same error coverage as DWC with 100% time redundancy but very little space redundancy. However, no one has demonstrated the practicality of double execution and RESO for online testing of general logic circuits.

A third, widely used form of redundancy is information redundancy—the addition of redundant coded information such as a parity-check bit.⁵ Such codes are particularly effective for detecting memory and data transmission errors, since memories and networks are susceptible to transient errors. Coding methods can also detect errors in data computed during critical operations.

Built-in self-test

For critical or highly available systems, a comprehensive online-testing approach that covers all expected permanent, intermittent, and transient faults is essential. In recent years, BIST has emerged as an important method of testing manufacturing faults, and researchers increasingly promote it for online testing as well.

BIST is a design-for-testability technique that places test functions physically on chip with the CUT, as illustrated in Figure 2. In normal operating mode, the CUT receives its inputs from other modules and performs the function for which it was de-

signed. In test mode, a test pattern generator circuit applies a sequence of test patterns to the CUT, and a response monitor evaluates the test responses. In the most common type of BIST, the response monitor compacts the test responses to form fault signatures. It compares the fault signatures with reference signatures generated or stored on chip, and an error signal indicates any discrepancies detected. We assume this type of BIST in the following discussion.

In developing a BIST methodology for embedded systems, we must consider four primary parameters related to those listed earlier for online-testing techniques:

- *fault coverage*—the fraction of faults of interest that the test patterns produced by the test generator can expose and the response monitor can detect. Most monitors produce a fault-free signature for some faulty response sequences, an undesirable property called aliasing.
- *test set size*—the number of test patterns produced by the test generator. Test set size is closely linked to fault coverage; generally, large test sets imply high fault coverage. However, for online testing, test set size must be small to reduce fault and error latency.
- *hardware overhead*—the extra hardware needed for BIST. In most embedded systems, high hardware overhead is not acceptable.
- *performance penalty*—the impact of BIST hardware on normal circuit performance, such as worst-case (critical) path delays. Overhead of this type is sometimes more important than hardware overhead.

System designers can use BIST for nonconcurrent, online testing of a system’s logic and memory.⁶ They can readily configure the BIST hardware for event-triggered testing, tying the BIST control to the system reset so that testing occurs during system start-up or shutdown. BIST can also be designed for periodic testing with low fault latency. This requires incorporating a test process that guarantees the detection of all target faults within a fixed time.

Designers usually implement online BIST with the goals of complete fault coverage and low fault latency. Hence, they generally design the test generator and the response monitor to guarantee coverage of specific fault models, minimum hardware overhead, and reasonable test set size. Different parts of the system meet these goals by different techniques.

Test generator and response monitor implementa-

tions often consist of simple, counterlike circuits, especially linear-feedback shift registers.⁵ An LFSR is formed from standard flip-flops, with outputs of selected flip-flops being fed back (modulo 2) to its inputs. When used as a test generator, an LFSR is set to cycle rapidly through a large number of its states. These states, whose choice and order depend on the LFSR’s design parameters, define the test patterns. In this mode of operation, an LFSR is a source of pseudorandom tests that are, in principle, applicable to any fault and circuit types. An LFSR can also serve as a response monitor by counting (in a special sense) the responses produced by the tests. After receiving a sequence of test responses, an LFSR response monitor forms a fault signature, which it compares to a known or generated good signature to determine whether a fault is present.

Ensuring that fault coverage is sufficiently high and the number of tests is sufficiently low are the main problems with random BIST methods. Researchers have proposed two general approaches to preserve the cost advantages of LFSRs while greatly shortening the generated test sequence. One approach is to insert test points in the CUT to improve controllability and observability. However, this approach can result in performance loss. Alternatively, one can introduce some determinism into the generated test sequence—for example, by inserting specific “seed tests” known to detect hard faults.

Some CUTs, including data path circuits, contain hard-to-detect faults that are detectable by only a few test patterns, denoted T_{hard} . An N -bit LFSR can generate a sequence that eventually includes $2^N - 1$ patterns (essentially all possibilities). However, the probability that the tests in T_{hard} will appear early in the sequence is low. In such cases, one can use deterministic testing, which tailors the generated test sequence to the CUT’s functional properties, instead of random testing. Deterministic testing is especially suited to RAMs, ROMs, and other highly regular components. A deterministic technique called transparent BIST³ applies BIST to RAMs while preserving the RAM contents—a particularly desirable feature for online testing. Keeping hardware overhead acceptably low is the main difficulty with deterministic BIST.

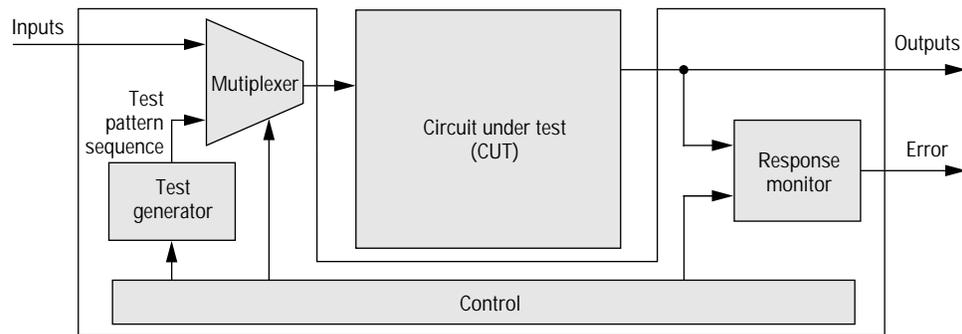


Figure 2. Generic BIST scheme.

A straightforward way to generate a specific test set is to store it in a ROM and address each stored test pattern with a counter. Unfortunately, ROMs tend to be much too expensive for storing entire test sequences. An alternative method is to synthesize a finite-state machine that directly generates the test set. However, the relatively large test set size and test vector width, as well as the test set's irregular structure, are much more than current FSM synthesis programs can handle.

Another group of test generator design methods, loosely called deterministic, attempt to embed a complete test set in a specific generated sequence. Again the generated tests must meet the coverage, overhead, and test size constraints we've discussed. An earlier article⁷ presents a representative BIST design method for data path circuits that meets these requirements. The test generator's structure, based on a twisted-ring counter, is tailored to produce a regular, deterministic test sequence of reasonable size. One can systematically rescale the test generator as the size of a non-bit-sliced data path CUT, such as a carry-look-ahead adder, changes.

Instead of using an LFSR, a straightforward way to compress test response data and produce a fault signature is to use an FSM or an accumulator. However, FSM hardware overhead and accumulator aliasing are difficult parameters to control. Keeping hardware overhead acceptably low and reducing aliasing are the main difficulties in response monitor design.

An example

As noted earlier, the control of passenger transportation systems—from automobiles to airplanes—is a major application for embedded real-time controllers. A high-end, late-model car may contain more than a hundred embedded microcontrollers, supervising both critical and high-availability functions including air bag deployment, antilock braking, engine throttle control, and collision avoidance. These applications require online testing with low error latency and very high fault coverage to ensure rapid detection of faults and fail-safe operation.

Automobile manufacturers use many of the previously mentioned online-testing techniques as well as a large variety of software checks to detect and handle faults and errors in such applications. For example, embedded controllers in several antilock braking systems use the classic DWC approach, in which two copies of the host microcontroller operate in a master-slave configuration with comparison at various checkpoints. A disagreement between processors invokes a software-controlled diagnostic or shutdown procedure. Although the DWC technique has low error detection latency with essentially no performance overhead, it requires more than 100% hardware overhead—

a high cost penalty for most automotive applications.

In an effort to reduce this cost, a European consortium involving the Robert Bosch Company, the University of Erlangen-Nuremberg, and others developed the AE11 microcontroller (see page 57).⁸ The AE11 is intended to replace dual controllers in critical applications, including automotive and railway controllers. It is a single-chip design using SGS-Thomson's CMOS5 technology with a 0.7-micron feature size and two metal layers. Compatible with Intel's 8051 eight-bit instruction-set architecture, the AE11 has a 4-Kbyte main memory (RAM) and a fairly conventional set of peripheral I/O modules. It is designed to detect and respond to hardware faults within milliseconds under all operating conditions. A supervisory system can respond either with a fail-safe shutdown or by gracefully degrading the system's performance.

The main design objectives for the AE11 were very high fault coverage and very short fault detection latency. To meet these stringent goals, the developers included essentially all the online-testing features discussed earlier—particularly concurrent checking based primarily on parity coding, and time-triggered checking based on special BIST circuitry. Because the fault detection latency is very short (a few milliseconds), the BIST functions must be invoked periodically (at least once every fault latency period). Thus, these functions belong in the online category.

Figure 3 outlines the AE11's structure and main test features. These include the following:

- parity code checking throughout the system
- parity checking and ALU parity prediction in the CPU data path
- program control flow checking via signature monitoring
- self-checking address-decoding logic in the RAM
- programmable watchdog timer
- pseudorandom and I_{DDQ} testing of peripheral modules
- power supply and temperature monitoring
- test control logic using boundary scan and a test access port controller

Since RAM occupies a large fraction of the total chip area, the AE11 designers devoted considerable attention to its online-testing needs. They added a parity-check bit to both the address and data buses and designed the memory's parity checkers to be concurrently self-checking.⁹ Additional self-testing logic detects both word line and address decoder faults. Special circuits detect bridging faults, including resistive shorts that result in arbitrary voltage levels in the decoder. The AE11's stringent fault latency requirements rule out most conventional RAM BIST methods, which are too slow and destroy stored data during testing.

The CPU design presents two testing problems: how to check the data path, including the ALU, and how to check

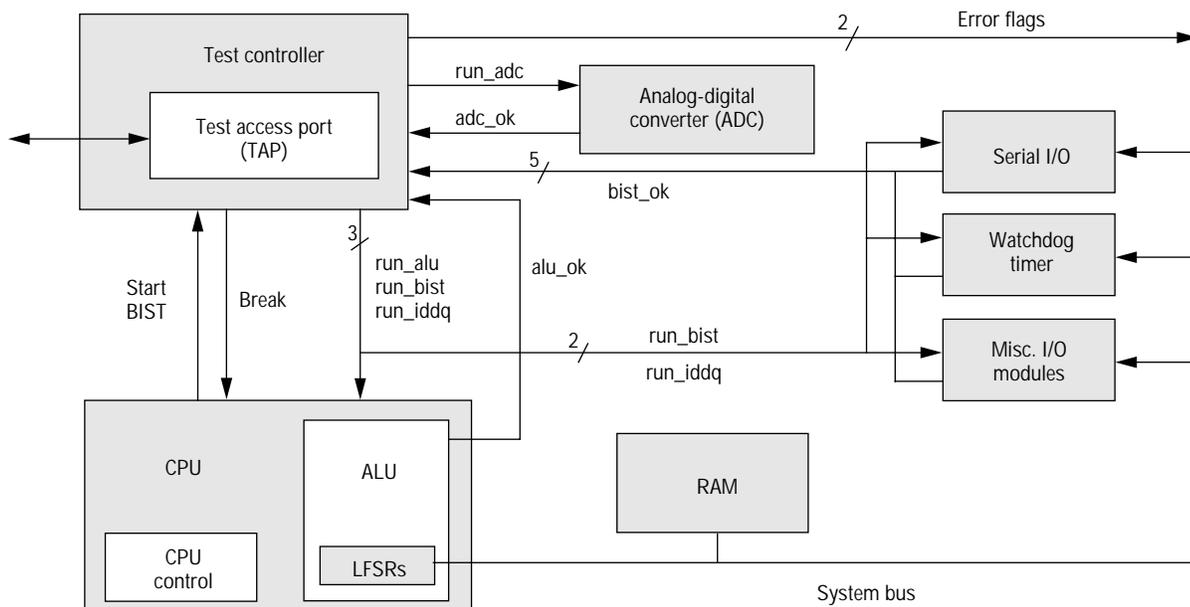


Figure 3. Key testing features of the AE11 fail-safe controller.

the control logic. The traditional approach to concurrent checking of a CPU's data path is to use an arithmetic code, such as a residue code, whose error-detecting property, unlike that of a parity code, is preserved by arithmetic operations. The AE11's designers used an alternative approach called parity prediction, which has the advantages of lower cost and compatibility with the AE11's overall use of parity codes. (Wakerly⁹ describes a contrasting self-checking microprocessor design that uses residue codes.)

The CPU's control unit also applies parity checking to control words. An interesting additional feature is the use of software signatures, codeword-like words computed during compilation of application programs and inserted automatically into the program flow. The AE11 executes special instructions that monitor these signatures to detect many types of hardware and software errors.

The designers used parity checking wherever the AE11's data path extends into the I/O subsystem. The AE11 uses on-line BIST and I_{DDQ} testing on the peripheral modules (Figure 3). A test controller conforming to the IEEE 1149.1 boundary-scan standard handles these test functions directly, with ultimate control of the test functions residing in software executed by the CPU. The BIST logic employs pseudorandom test and signature generation implemented by LFSRs in the CPU and multiple-input signature registers (MISRs) in the peripheral modules. The use of many pseudorandom test patterns provides a measure of protection against unknown or nontargeted hardware faults, as well as standard stuck-at faults. The system data bus transmits the pseudo-

random test vectors from the LFSRs to the circuits being tested. The MISRs compress the resulting responses to obtain signatures that control the *bist_ok* signals.

Some of the test vectors also set up I_{DDQ} tests that use on-chip current monitors. The AE11 also includes special fault detection and error-monitoring procedures for components such as the analog-to-digital converter (ADC). Other specialized test circuits monitor the power supply voltage and the chip temperature.

The following is the AE11's overall test process: At system start-up, the test controller executes a set of tests to check that the major subsystems—CPU, RAM, and peripheral modules—are fault-free. During normal operation, the concurrent-checking circuits flag errors as soon as they occur. In addition, the microcontroller stops periodically (under software-controlled periods depending on the acceptable fault latency) to execute I_{DDQ} and various functional tests. Functional tests are needed to test circuits that are not self-testing, such as the interrupt controller and the ADC's input channels.

The developers estimate that the AE11's self-testing hardware features achieve over 99.7% coverage of modeled faults and errors, with less than 35% area overhead and less than 15% performance loss.

It is interesting to compare this design with Wakerly's early work,⁹ as well as Nicolaidis's more recent design of a self-checking version of the Motorola 68000 microprocessor.¹⁰ The AE11 is an intriguing case of the application of hardware features for hardware self-testing in a single, low-cost

microcontroller. However, its ultimate usefulness in system design remains to be demonstrated. There are reasons for duplicating functions off chip besides the detection of hardware faults, including online software testing, diagnosis, and fault tolerance. Nevertheless, the AE11 demonstrates that most major online-testing techniques, including BIST, can be implemented simultaneously as on-chip hardware components at moderate cost. Thus, these techniques can serve as basic building blocks in embedded-system design. Self-testing microcontrollers like the AE11 permit the design of critical systems without external monitoring. Therefore, as critical systems proliferate, sophisticated microcontrollers of this kind will become essential components.

ONLINE TESTING IS FAST BECOMING a basic feature of embedded systems, not only for critical applications, but also to meet the high-availability requirements of common consumer products. To achieve the twin goals of high fault and error coverage and low error latency, the system must include advanced hardware and software features for testing and monitoring. One such hardware feature is BIST, a technique widely applied in manufacturing testing and widely promoted for online testing. The type of online testing for which BIST is most appropriate is nonconcurrent testing for permanent and intermittent faults. But BIST can be only part of a comprehensive approach to online testing. We need further research to learn how to systematically integrate hardware and software to enhance safety and reliability in embedded systems. 

Acknowledgments

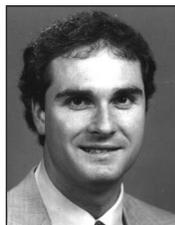
General Motors Global Research and Development Operations supported this research.

References

1. M.R. Lyu, ed., *Software Fault Tolerance*, John Wiley & Sons, New York, 1995.
2. K.K. Saluja, R. Sharma, and C.R. Kime, "A Concurrent Testing Technique for Digital Circuits," *IEEE Trans. Computer-Aided Design*, Vol. 7, No. 12, Dec. 1988, pp. 1250-1259.
3. M. Nicolaidis, "Theory of Transparent BIST for RAMs," *IEEE Trans. Computers*, Vol. 45, No. 10, Oct. 1996, pp. 1141-1156.
4. A. Mahmood and E. McCluskey, "Concurrent Error Detection Using Watchdog Processors—A Survey," *IEEE Trans. Computers*, Vol. 37, No. 2, Feb. 1988, pp. 160-174.
5. B.W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, Mass., 1989.
6. B.T. Murray and J.P. Hayes, "Testing ICs: Getting to the Core of the Problem," *Computer*, Vol. 29, No. 11, Nov. 1996, pp. 32-45.
7. H. Al-Asaad, J.P. Hayes, and B.T. Murray, "Scalable Test Generators for High-Speed Data Path Circuits," *J. Electronic Testing: Theory and Applications*, Vol. 12, No. 1/2, Feb./Apr. 1998, pp. 111-125 (reprinted in *On-Line Testing for VLSI*, M. Nicolaidis, Y. Zorian, and D.K. Pradhan, eds., Kluwer, Boston, 1998).
8. E. Böhl, T. Lindenkrenz, and R. Stephan, "The Fail-Stop Controller AE11," *Proc. Int'l Test Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1997, pp. 567-577.
9. J. Wakerly, *Error Detecting Codes, Self-Checking Circuits, and Applications*, North-Holland, New York, 1978.
10. M. Nicolaidis, "Efficient UBIST Implementation for Microprocessor Sequencing Parts," *J. Electronic Testing: Theory and Applications*, Vol. 6, No. 3, June 1995, pp. 295-312.



Hussain Al-Asaad is an assistant professor in the department of Electrical and Computer Engineering at the University of California, Davis. His research interests include design verification, testing, and fault-tolerant computing. Al-Asaad received his BE in computer and communications engineering from the American University of Beirut, Lebanon, his MS in computer engineering from Northeastern University, Boston, and his PhD in computer science and engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE, ACM, and Sigma Xi.



Brian T. Murray is a member of the technical staff of General Motors Global Research and Development Operations, where he has led projects in testing, high-dependability embedded systems, and computer architecture. He is also an adjunct lecturer at the University of Michigan. Murray received the AB in physics and mathematics from Albion College, the MS in electrical engineering from Duke University, and the PhD in computer science and engineering from the University of Michigan. He is a member of the IEEE, ACM, Sigma Xi, and Phi Beta Kappa.



John P. Hayes is a professor in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor. He teaches and conducts research in computer-aided design verification and testing, computer architecture, VLSI design, and fault-tolerant computing. Hayes received the BE from the National University of Ireland, Dublin, and the MS and PhD from the University of Illinois, all in electrical engineering. He is an IEEE fellow and a member of ACM and Sigma Xi.

Send questions and comments about this article to Hussain Al-Asaad, University of California, Davis, Dept. of Electrical and Computer Engineering, One Shields Ave., Davis, CA 95616-5294; halasaad@ece.ucdavis.edu.