ON-LINE BUILT-IN SELF-TEST FOR OPERATIONAL FAULTS

Hussain Al-Asaad and Mayank Shringi
Department of Electrical & Computer Engineering
University of California
One Shields Avenue, Davis, CA 95616-5294
Tel: (530) 792-7002
E-mail: halasaad@ece.ucdavis.edu

***Abstract*—On-line testing is fast becoming a basic feature of digital systems, not only for critical applications, but also for highly-available applications. To achieve the goals of high error coverage and low error latency, advanced hardware features for testing and monitoring must be included. One such hardware feature is built-in self-test (BIST), a technique widely applied in manufacturing testing.**

We present a practical on-line periodic BIST method for the detection of operational faults in digital systems. The method applies a near-minimal deterministic test sequence periodically to the circuit under test (CUT) and checks the CUT responses to detect the existence of operational faults. To reduce the testing time, the test sequence may be partitioned into small sequences that are applied separately—this is especially useful for real-time digital systems. Several analytical and experimental results show that the proposed method is characterized by full error coverage, bounded error latency, moderate space and time redundancy.

1. Introduction

The field of digital systems has undergone a major revolution in recent decades. Circuits are shrinking in physical size while growing both in speed and range of capabilities. This rapid advancement is not without serious problems, however. Especially worrisome are verification and testing, which become more important as the system complexity increases and time-to-market decreases. The inadequacy of existing verification methods is illustrated by the 1994 Pentium microprocessor's FDIV design error, which cost its manufacturer (Intel) an estimated $500 million. The inadequacy of existing testing methods is also illustrated by the 1990 breakdown of AT&T's long distance network, which cost AT&T around $75 million.

Due to the high cost of failure, verification and testing now account for more than half of the total lifetime cost of an integrated circuit (IC). Increasing emphasis needs to be placed on finding design errors and physical faults as early as possible in the life of a digital system, new algorithms need to be devised to create tests for logic circuits, and more attention should be paid to synthesis for test and on-line testing. On-line testing requires embedding logic that continuously checks the system for correct operation. Built-in self-test (BIST) is a technique that modifies the IC by embedding test mechanisms directly into it. BIST is often used to detect faults before the system is shipped and is potentially a very efficient way to implement on-line testing.

This paper focuses on on-line testing of digital hardware using BIST. Section 2 discusses on-line testing for digital systems [1], including what to test for, and how and when to test; it also gives a taxonomy of on-line hardware testing methods. In Section 3, we review the most common BIST method, which is typically used to test for manufacturing faults, and discuss how it can be applied to on-line testing. We then present a detailed case study in Section 4.

2. On-Line Testing

Faults are physical or logical defects in the design or implementation of a device. Under certain conditions, they lead to *errors*, that is, incorrect system states. Errors induce failures, that is, a deviation from appropriate system behavior. If the failure can lead to an accident, it is a *hazard.* Faults throughout the life of a digital system can be classified into three groups: design, fabrication, and operational faults. Design faults are made by human designers or CAD software (simulators, translators, or layout generators), and occur during the design process. Fabrication defects result from an imperfect manufacturing process. Operational faults are caused by wearout or environmental disturbances during normal operation of the digital system. Such disturbances include electromagnetic interference, operator mistakes, and extremes of temperature and vibration. Some design defects and manufacturing faults escape detection and combine with wearout and environmental disturbances

to cause problems in the field.

Operational faults are usually classified according to their duration:

- *Permanent* faults remain in existence indefinitely if no corrective action is taken. Many of these are residual design or manufacturing faults. Those that are not most frequently occur during changes in system operation, for instance, after system start-up or shutdown, or as a result of a catastrophic environmental disturbance such as a collision involving the product that contains the digital system.
- *Intermittent* faults appear, disappear, and reappear repeatedly. They are difficult to predict, but their effects are highly correlated. Most intermittent faults are due to marginal design or manufacturing steps. 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.

On-line testing addresses the detection of operational faults, and is found in computers that support critical or high-availability applications. The goal of on-line testing is to detect fault effects, that is, errors, and take appropriate corrective action. For example, in some critical applications, the system is shut down after an error is detected. In other applications, error detection triggers a reconfiguration mechanism that allows the system to continue its operation, perhaps with some degradation in performance. On-line testing can be performed by external or internal monitoring, using either hardware or software; internal monitoring is referred to as *self-testing*. Monitoring is internal if it takes place on the same substrate as the circuit under test (CUT); nowadays, this usually means inside a single IC—a system-on-a-chip (SOC).

There are four primary parameters to consider in the design of an on-line testing scheme:

- *Error coverage* (*EC*): This is defined as the fraction of all modeled errors that are detected, usually expressed in percent. Critical and highly available systems require very good error detection or *error coverage* to minimize the impact of errors that lead to system failure.
- *Error latency* (*EL*): This is the difference between the first time the error is activated and the first time it is detected. *EL* is affected by the time taken to perform a test and by how often tests are executed. A related parameter is *fault latency* (*FL*), defined as the difference between the onset of the fault and its detection. Clearly, $FL \geq EL$, so when *EL* is difficult to determine, *FL* is often used instead.
- *Space redundancy* (*SR*): This is the extra hardware or firmware needed to perform on-line testing.
- *Time redundancy* (*TR*): This is the extra time needed to perform on-line testing.

An ideal on-line 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 the CUT. Most on-line testing methods meet some of these constraints without addressing others. Consideration of all the parameters discussed above in the design of an on-line testing scheme can create conflicting goals. High coverage can require high *EL*, *SR* and/or *TR*. Schemes with immediate detection ($EL = 1$) minimize time redundancy, but require more hardware. On the other hand, schemes with delayed detection ($EL > 1$) reduce the time and space redundancy at the expense of increased error latency. Several proposed on-line testing techniques that use delayed detection assume equiprobable input combinations and try to establish a probabilistic bound on the error latency [10]. This results in certain faults remaining undetected for a long time because tests for them rarely if ever appear at the inputs of the CUT.

To cover all of the fault types described earlier, two different modes of on-line testing are employed: *concurrent testing* which takes place during normal system operation, and *non-concurrent testing* which takes place while normal operation is temporarily suspended. These operating modes must often be overlapped to provide a comprehensive on-line testing strategy at acceptable cost.

Non-concurrent 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 in the life of a system, such as start-up or shutdown, and its goal is to detect permanent faults. It is usually advisable to detect and repair permanent faults as soon as possible. Event-triggered tests resemble manufacturing tests. Any such test can be applied on-line, as long as the required testing resources are available. Typically the hardware is partitioned into components, each of which is exercised by tests specific to that component.

Time-triggered testing is activated at predetermined

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

Concurrent testing: Non-concurrent 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 by itself particularly useful for diagnosing the source of errors, so it is often combined with diagnostic software. It may also be combined with non-concurrent 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 [7]. This is a counter that must be reset by the system on a repetitive basis to indicate that the system is functioning properly. A watchdog timer is based on the assumption that the system is fault-free—or at least alive—if it is able to perform the simple task of resetting the timer at appropriate intervals, which implies that control flow is correctly traversing timer reset points. Proper system sequencing can be monitored to very high precision by guarding watchdog timer reset operations with software-based acceptance tests that check signatures computed while control flow traverses various checkpoints. More complex hardware watchdogs can be constructed that implement this last approach in hardware [7].

A key element of concurrent testing for data errors is redundancy. For example, *duplication with comparison* (DWC) [5] can detect any single error at the expense of 100% space redundancy. DWC requires two copies of the CUT, which operate in tandem with identical inputs. Their outputs are compared and any discrepancy indicates an error. In many applications, the high hardware overhead of DWC is unacceptable. Moreover, it is difficult to prevent minor variations in timing between duplicated modules from invalidating comparisons. A possible lower-cost alternative is time redundancy. Critical operations can be executed more than once, at diverse time points, and their results compared. This technique is called *double-execution* or *retry*. Transient faults are likely to affect only one instance of the operation and can thus be detected. *Recomputing with shifted operands* (RESO) [5] achieves almost the same error coverage of DWC with 100% time redundancy but very little space redundancy. However, the practicality of double execution and RESO in on-line testing for general logic circuits has not been demonstrated. A third form of redundancy which is very widely used is information redundancy, that is, 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, however, coding methods can also detect errors in data computed during critical operations.

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

3. On-Line Deterministic Built-in Self-Test

BIST is a design-for-testability technique that places the testing functions physically with the CUT, as illustrated in Figure 1. In normal operating mode, the CUT receives its inputs *X* from other modules and performs the function for which it was designed. In test mode, a test pattern generator circuit TG applies a sequence of test patterns *S* to the CUT, and the test responses are evaluated by a response monitor RM. In the most common type of BIST, test responses are compacted in RM to form (fault) *signatures.* The response signatures are compared with reference signatures generated or stored on-chip, and the error signal indicates any discrepancies detected.

Four primary parameters must be considered in developing a BIST methodology for digital systems; these correspond with the design parameters for on-line test-

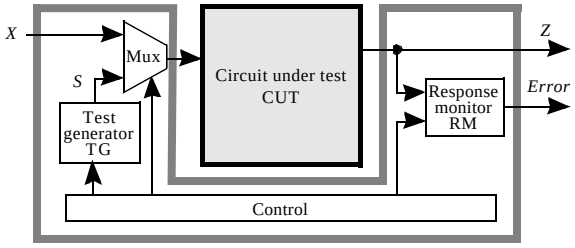

Figure 1 Generic BIST scheme.

ing techniques discussed earlier in Section 2.

- *Fault coverage*: This is the fraction of faults of interest that can be exposed by the test patterns produced by TG and detected by RM.
- *Test set size*: This is the number of test patterns produced by the TG, and is closely linked to fault coverage: generally, large test sets imply high fault coverage. However, for on-line testing, test set size must be kept small to reduce *FL* and *EL*.
- *Hardware overhead*: The extra hardware needed for BIST is considered to be overhead. In most digital systems, high hardware overhead is not acceptable, as discussed earlier.
- *Performance penalty*: This refers to the impact of BIST hardware on normal circuit performance such as its worst-case (critical) path delays. Overhead of this type is sometimes more important than hardware overhead.

BIST can be used for non-concurrent, on-line testing of the logic and memory parts of a system [8][9]. It can readily be configured for event-triggered testing, in which case, the BIST control can be tied 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 testing process into the CUT that guarantees the detection of all target faults within a fixed time. We next describe how periodic BIST is used for combinational and sequential circuits.

3.1 Combinational Circuits

On-line BIST is usually implemented with the twin goals of complete fault coverage and low fault latency. Hence, the TG and RM are generally designed to guarantee coverage of specific fault models, minimum hardware overhead, and reasonable test set size. Consequently, the use of deterministic tests is often the most attractive option for on-line testing.

We model an operational fault as $OF(f, D)$, where f is a stuck-at fault and D is the duration of existence of f. For example, a permanent operational fault is modeled as $OF(f, \infty)$, which is equivalent to the classical stuck-at fault f. The duration of intermittent and transient operational faults is assumed to be large enough for these faults to be detected. In the rest of this paper, we assume that operational faults are modeled as stuck-at faults with large duration of existence.

We now analyze on-line deterministic BIST for combinational circuits. Assume that we have a complete test set $T = \{t_1, t_2, ..., t_q\}$ for the set of stuck-at faults $F = \{f_1, f_2, ..., f_m\}$. Testing is overlapped with normal operation as shown in Figure 2. Since every period P is divided into k clocks of normal operation and l clocks of testing, then the time redundancy of this on-line deterministic BIST scheme is l/k. In our analysis, we consider the case of $l = 1$, where time redundancy is minimum.

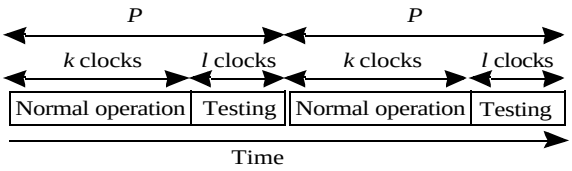

Figure 2 Overlapping of testing and normal operation on the CUT.

On-line deterministic BIST has low hardware overhead which can be as small as 5% of the CUT as we show later in Section 4. It has little time redundancy and a deterministic bound on the worst case fault latency. It can achieve 100% error coverage with little design effort. On-line deterministic BIST guarantees the detection of errors within a given period of time as shown by the following lemma.

Lemma 1 *If the test set T is minimal, then the worst case fault latency L_{max} is $1 + (k + 1)|T|$.*

Proof: If T is minimal, then there exist a fault f_i that is detected by exactly one test t_j. If this is not the case, then if we remove a test from T, no fault will become undetectable by T, which contradicts the minimality of T. If f_i occurs in the same clock where test t_j is applied, then f_i will not be detected until t_j is applied again. Hence, the fault latency for f_i is $1 + (k + 1)|T|$. □

The above lemma gives the worst case fault latency, however, the average fault latency is well below $1 + (k + 1)|T|$. This is due to the fact that most faults in F are detected by more than one test of T. For the case of stuck-at faults, an experiment was conducted to determine the average number of tests detecting a given fault from the set of stuck-at faults F using a minimal complete test set for F generated by the ATPG tool ATALANTA [6]. It turned out that in the ISCAS-85 benchmark circuits [4], an average of 20% of the tests detect an arbitrary fault of F. Also, we found that there are faults in F that are detected by most tests of T. A rough estimate of the fault latency that is independent of the order of applying the tests is given by,

$$L_{avg} = \frac{1 + (k + 1)|T|}{r} = \frac{|T|}{r}k + \frac{1 + |T|}{r} = \sigma k + \rho$$

where r is the average number of tests detecting a given

Table 1 Statistics about deterministic test sets for stuck-at faults and the average fault latency in on-line deterministic BIST.

Circuit	Test set size $\|T\|$	Fault set size $\|F\|$	No. of tests detecting a given fault		$L_{avg} = \sigma k + \rho$	
			Average (r)	Max	σ	ρ
c17	5	22	1.82	4	2.75	3.30
c432	46	520	5.41	42	8.50	8.69
c499	52	750	18.62	51	2.80	2.85
c880	47	942	9.33	45	5.04	5.15
c1355	85	1566	23.25	83	3.66	3.70
c1908	115	1870	27.47	113	4.19	4.22
c2670	106	2630	21.87	106	4.85	4.90
c3540	152	3291	20.96	149	7.25	7.30
c5315	106	5291	17.35	105	6.11	6.17
c6288	35	7710	12.37	32	2.83	2.91
c7552	199	7418	34.97	198	5.69	5.72

fault. The results obtained using the above equation are shown in Table 1.

The actual average fault latency is dependent on the order of tests during test application. So, we can get smaller fault latency by reordering the tests in the test sequence. Trying to find the best order of tests in the test sequence out of the possible $(|T| - 1)!$ orderings is computationally expensive and is often impossible. Therefore, heuristics need to be developed to determine a near-optimal ordering. In addition to ordering the tests, we conjecture that repeating certain tests from T reduces the fault latency. If the tests that detect large number of faults are repeated, then the average fault latency will decrease, however, the worst case latency may increase.

3.2 Sequential Circuits

We now analyze on-line deterministic BIST for sequential circuits. Assume that we have a complete test sequence $S = \{t_1, t_2, ..., t_q\}$ for the set of stuck-at faults $F = \{f_1, f_2, ..., f_m\}$. Testing is overlapped with normal operation similar to the case of combinational circuits. Since every period is divided into k clocks of normal operation and l clocks of testing, then the time redundancy of this on-line deterministic BIST scheme is l/k. Since partitioning of the test sequence may not be possible in some cases, we consider the case of $l = q$, where the total test sequence is applied in one burst. On-line deterministic BIST guarantees the detection of errors within a given period of time. In fact the worst case fault latency L_{max} is $2l + k$. However, the average fault latency is well below the above limit. An experiment was conducted to determine the average fault latency for the ISCAS-89 benchmark circuit s27 [3]. We first generated a complete test set sequence of 20 test vectors that detect all 32 stuck-at faults in the s27 circuit using a sequential ATPG tool. With $l = 20$, the worst case fault latency is $k + 40$. We then computed the average fault latency for every stuck-at fault and then averaged the result over all faults. We assume that a fault have equal probability of occurrence in any clock cycle that is independent of the mode of operation (testing or normal). The average fault latency for the s27 benchmark circuit was computed manually as follows:

$$L_{avg}(s27) = \frac{k^2 + 54.68k + 688.75}{2k + 40}$$

The above equation implies that the average fault latency for s27 is approximately one-half of the maximum fault latency.

The above discussion of on-line deterministic BIST demonstrate that it can achieve 100% error coverage, deterministically bounded error latency, and small time redundancy. Moreover, the performance penalty due to BIST is limited to the extra delay of the multiplexers at the inputs of the CUT. The area overhead of on-line deterministic BIST can be made relatively small as we show next in the case study.

4. Case Study—Multiply-Add Unit

In this section, we apply our on-line deterministic BIST method to a multiply-add unit (MAU). The high-level model and some implementation details of the target $n \times n$-bit MAU are shown in Figure 3 [2]. The MAU is composed of a cascaded sequence of carry-save adders followed by a carry-lookahead adder in the last stage. This design is scalable and faster than a normal multiply-add unit where the last stage is a ripple-carry adder. Table 2 shows a possible test sequence of size 20 for 4×4-bit MAU [2]. This test sequence can be easily extended to $n \times n$-bit MAU, resulting in a test set of size $4n + 4$.

A scalable test generator TG for $n \times n$-bit MAU is shown in Figure 4. A scalable response monitor RM for $n \times n$-bit MAU is shown in Figure 5. The BIST hardware overhead for various values of n for the MAU is shown in Table 3. The hardware overhead of the overall BIST is less than 10% for a 32×32-bit MAU.

From the above case study, we can conclude that on-line deterministic BIST for the MAU guarantees 100% coverage of stuck-at faults, bounded error latency, low hardware overhead, low time redundancy, and negligible performance penalty.

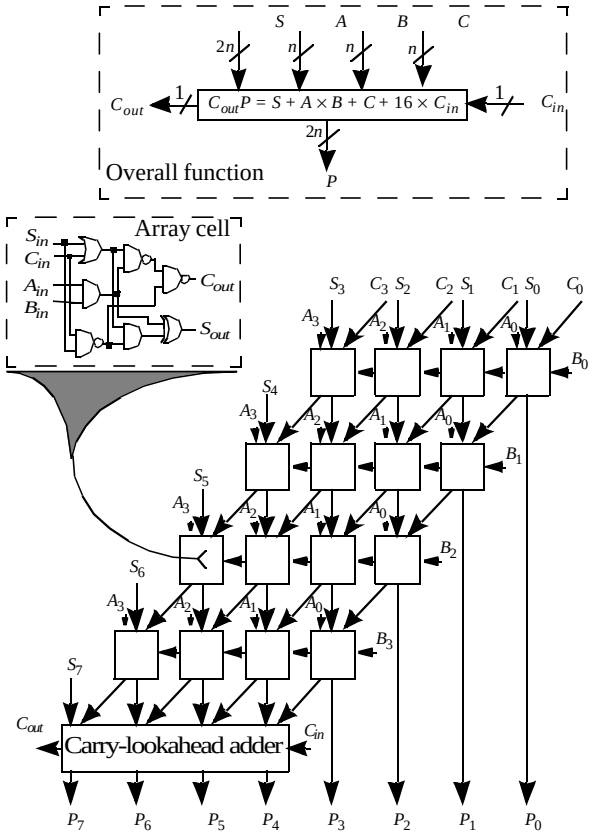

Figure 3 High-level model for the multiply-add unit.

Table 2 Complete and near-minimal scalable test sequence for the multiply-add unit.

Test #	$A_3B_3C_3S_7S_3$	$A_2B_2C_2S_6S_2$	$A_1B_1C_1S_5S_1$	$A_0B_0C_0S_4S_0$	C_{in}
1	11100	11100	11100	11100	1
2	11100	11100	11100	11000	1
3	11100	11100	11000	11101	1
4	11100	11000	11101	11101	1
5	11000	11101	11101	11101	1
6	00011	00011	00011	00011	0
7	00011	00011	00011	00111	0
8	00011	00011	00111	00010	0
9	00011	00111	00010	00010	0
10	00111	00010	00010	00010	0
11	10100	10100	10100	10100	1
12	10100	10100	10100	10000	1
13	10100	10100	10000	10101	1
14	10100	10000	10101	10101	1
15	10000	10101	10101	10101	1
16	01011	01011	01011	01011	0
17	01011	01011	01011	01111	0
18	01011	01011	01111	01010	0
19	01011	01111	01010	01010	0
20	01111	01010	01010	01010	0

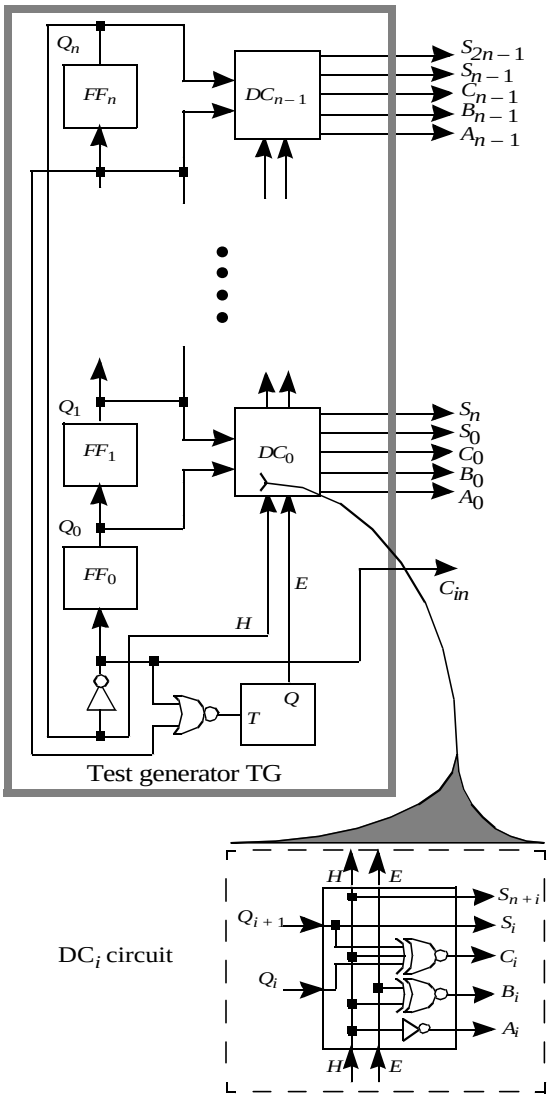

Figure 4 Test generator for the $n \times n$-bit multiply-add unit.

Table 3 Hardware overhead for BIST in the $n \times n$-bit multiply-add unit.

Circuit module	Hardware overhead %					
	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 48$	$n = 64$
TG	18.6	9.1	4.5	2.3	1.5	1.1
RM	10.3	4.4	1.9	0.9	0.6	0.4
Overall BIST	66.0	34.5	17.8	9.1	6.1	4.6

5. Discussion

We have presented on-line deterministic periodic BIST and demonstrated that it is attractive approach to detect operational faults that arise in the field. We have also

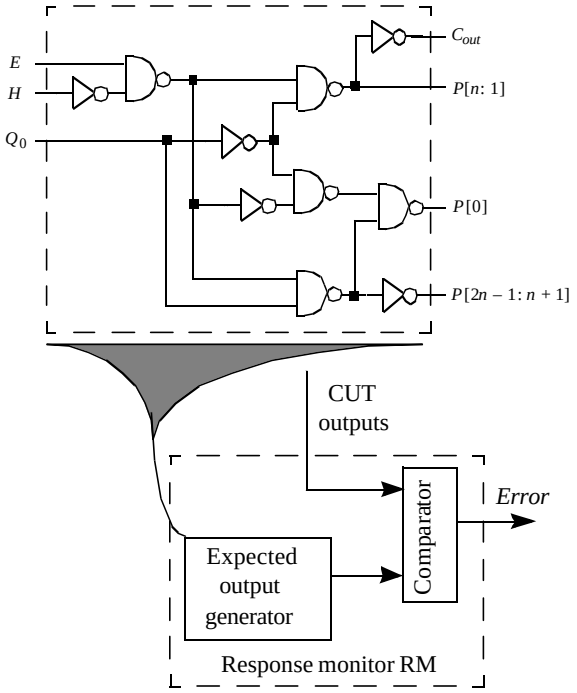

Figure 5 Response Monitor for the $n \times n$-bit multiply-add unit.

presented a detailed case study that shows the efficiency and feasibility of this approach. We have achieved 100% fault coverage, bounded error latency, low hardware overhead, and small time redundancy. The time redundancy of on-line BIST can be reduced by applying the tests to CUT while it is not being used. In this case, the testing becomes aperiodic. If several components of a digital systems are not fully utilized, aperiodic testing becomes more attractive. An application where utilization may be very low is embedded controllers—a typical embedded controller takes no action if the outside sensors' values remain unchanged. In such application, the controller can be tested during the idle time in the system. Pipeline structures is another example where aperiodic deterministic BIST is applicable. For this case, an idle detector need to be designed to watch the system and determine if the pipeline is not being used in a given clock cycle. If that happens, then a test is applied to the pipeline and the result is picked up after l clocks where l is the number of stages in the pipeline. In safety critical applications that demand an upper bound on the error latency, waiting for idle times may not be a good option. Instead, the system may be briefly stopped from normal operation to perform the required tests. Our future research is centered on exploring on-line aperiodic and deterministic testing in complex high-level designs such as microprocessors.

Acknowledgments

The research presented in this paper was supported by the Office of Vice Chancellor for Research at the University of California, Davis.

References

[1] H. Al-Asaad, B. T. Murray, and J. P. Hayes, "Online BIST for embedded systems", *IEEE Design and Test of Computers*, Vol. 15, No. 4, pp. 17-24, November 1998.

[2] H. Al-Asaad, J. P. Hayes, and B. T. Murray, "Scalable test generators for high-speed datapath circuits", *Journal of Electronic Testing: Theory and Applications*, Vol. 12, Nos. 1/2, pp. 111-125, February/April 1998. (Reprinted in M. Nicolaidis, Y. Zorian, and D. K. Pradhan (editors), *On Line-Testing for VLSI*, Kluwer, Boston, 1998.)

[3] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits", *Proc. International Symposium on Circuits and Systems*, 1989, pp. 1929-1934.

[4] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran", *Proc. International Symposium on Circuits and Systems*, 1985, pp. 695-698.

[5] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, Massachusetts, 1989.

[6] H. K. Lee and D. S. Ha, "On the generation of test patterns for combinational circuits", Dept. of Electrical Engineering, Virginia Polytechnic Institute and State University, Tech. Rep. 12-93, 1993.

[7] A. Mahmood and E. McCluskey, "Concurrent error detection using watchdog processors—A survey", *IEEE Transactions on Computers*, Vol. C-37, pp. 160-174, February 1988.

[8] B. T. Murray and J. P. Hayes, "Testing ICs: Getting to the core of the problem", *IEEE Computer*, Vol. 29, pp. 32-45, November 1996.

[9] M. Nicolaidis, "Theory of transparent BIST for RAMs", *IEEE Transactions on Computers*, Vol. 45, pp. 1141-1156, October 1996.

[10] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits", *IEEE Transactions on Computer-Aided Design*, Vol. 7, pp. 1250-1259, December 1988.