VLSI Design
1993, Vol. 1, No. 1, pp. 9–22
Reprints available directly from the publisher
Photocopying permitted by license only

© 1993 Gordon and Breach Science Publishers S.A.
Printed in the United States of America

Theory, Analysis and Implementation of an On-Line BIST Technique

RAJIV SHARMA*
Test Products Group, Cadence Design Systems Inc., Lowell, Massachusetts, USA

KEWAL K. SALUJA
Department of Electrical and Computer Engineering, University of Wisconsin, Madison, Wisconsin, USA

A *Built-In Concurrent Self-Test* (BICST) technique for testing combinational logic circuits concurrently with their normal operation is proposed. Concept of sharing the test hardware between identical circuits to reduce the overall area overhead is introduced. The method was implemented in the design of an ALU with on-line test capability in CMOS technology. The additional hardware used for a 12-bit ALU was 19% of the total chip area and it did not impose any timing overhead on the operation of the ALU. The overhead decreases with an increase in the size of the ALU.

Following the description of the BICST technique, measures for evaluating the performance of the BICST technique are defined. Methods for the computation of the performance measures using analytical and simulation techniques are discussed and results of these methods are reported. Methods for detecting intermittent faults and for computing the transient fault coverage using BICST are also described. The impact of BICST on the system diagnostics and system maintenance is discussed.

Key Words: *Concurrent testing; Test latency; Built-In self test; VLSI Testing; Transient faults; Intermittent faults*

Developments in the VLSI technology responsible for increase in the complexity of logic in integrated circuits are ever compounding the problem of testing digital circuits and systems. The testing is done at different stages of manufacturing and hence it is a continual problem as explained below. A device must be thoroughly tested before it is shipped. A device must also undergo an acceptance test before it is integrated into a system. After the initial acceptance test, periodic testing of integrated circuit chips at the system level has to be done to prevent the accumulation of faults. Such periodic testing is generally carried out off-line, that is the system is stopped and tested using a set of test vectors. Traditional techniques for off-line testing use algorithms to find a set of test vectors to detect the modeled faults in the circuit. These test vectors can either be applied by an external tester or they can be stored on chip and applied during test mode. The latter technique can be viewed as an off-line *Built-In Self-Test* (BIST) technique [1].

Almost all off-line BIST techniques employ extra hardware. This extra hardware can also be used for off-line maintenance tests of a system by periodically stopping the system for test purposes. Off-line periodic testing degrades overall system performance since the system becomes unavailable during these off-line maintenance tests. An alternative approach is to carry out some form of on-line or concurrent testing. That is, testing while the system is actually carrying out a useful computation.

In this paper we propose a novel technique for on-line testing, which we call *Built-In Concurrent-Self-Test* (BICST). BICST assumes the presence of underlying BIST resources for off-line testing. These resources are modified in such a way that they can be used for both off-line and on-line testing. By carrying out testing concurrently with the normal operation of a circuit/system, we shall show that BICST circumvents the performance degradation caused by periodic maintenance testing. We shall also show that

*This work was completed when the author was with UW-Madison.

BICST can provide the circuit with enhanced diagnostic capability, reduced system maintenance requirements, and detection capability for transient and intermittent faults in addition to permanent faults. Note that the off-line testing capability of the BIST resources must still be maintained for production testing purposes. We also propose a technique for sharing the BICST hardware resources between identical circuits, thereby reducing the overall extra overhead for testing.

BICST CONCEPT AND ARCHITECTURE

As mentioned in the introduction, conventional testing methods apply set of test vectors to the circuits that is undergoing an acceptance test. In the BICST technique proposed in this paper, we make use of the same test vectors to accomplish concurrent (online) testing of the circuit. However, the technique proposed in this paper is not restricted and limited to using such tests. Any test set which achieves the desired fault coverage and is suitable for BIST application can be used.

The block diagram of the BICST architecture is shown in Figure 1. Note that this figure and the details in this section are for the purpose of concept illustration only. Issues such as overhead, performance penalty, are the subjects of the subsequent sections. The *n-input, m-output* circuit to be tested during the normal operation is called *Circuit Under Test* (CUT). The extra hardware for BICST consists of a *Concurrent Test Circuit* (CTC) and a comparator. The normal inputs to the CUT are also fed to the CTC. The CTC has n inputs and m outputs (the same as for the CUT). The CTC is designed such that its outputs for those normal inputs to the CUT which are also test vectors for the CUT are the same as the expected response of the CUT. During normal operation of the circuit, occurrence of an input which is also a test for the CUT is termed as a HIT condition. The CTC generates a signal known as *Comparator Enable* (CE), for enabling the comparator, whenever a HIT occurs. Thus on every HIT the CUT outputs are compared with the CTC outputs, which are expected to be identical if the CUT is fault-free. In this way testing of the CUT proceeds concurrently with its normal operation and the testing of the CUT is said to be complete when all the test vectors necessary to test the CUT have appeared at least once as normal inputs to the CUT. The CTC marks the completion of the test by asserting the *Test Complete*

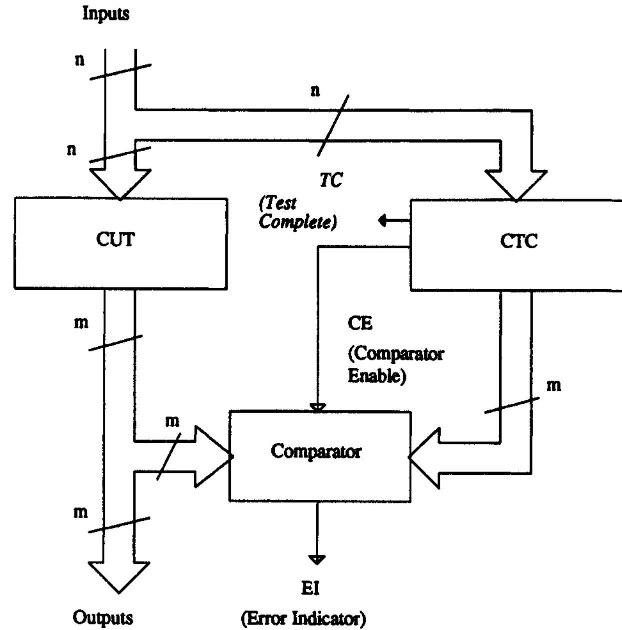


FIGURE 1 BICST Architecture.

(TC) signal. The CTC can be as simple a structure as a linear feedback shift register as explained in Saluja et al. [2] or as complex as an extra copy of the CUT.

We briefly digress from our discussion and describe the design of a CTC using a PLA. Let us assume that the *n-input, m-output* CUT can be tested by t test vectors. The PLA used to design the CTC for this CUT will consist of n input lines, m output lines, and t product lines. The PLA is programmed so that for each *n-bit* test vector the PLA output is the same as the expected response of the CUT for that test vector. Thus each product line of the PLA corresponds to a unique test vector. We illustrate the organization of the PLA by the following example. Consider a CUT with $n = 4$, $m = 6$, and $t = 5$. The five test vectors and their expected responses for the CUT are shown in Table I. The realization of the PLA is shown in Figure 2. The PLA has been augmented to generate the CE and the TC signals. A tag bit is attached to each product line of the PLA

TABLE I
Test/Response Table for PLA of the Example CTC

Test Vector	Expected Response
t_1 = 0010	100101
t_2 = 0011	011000
t_3 = 1011	100111
t_4 = 1100	111010
t_5 = 1101	101011

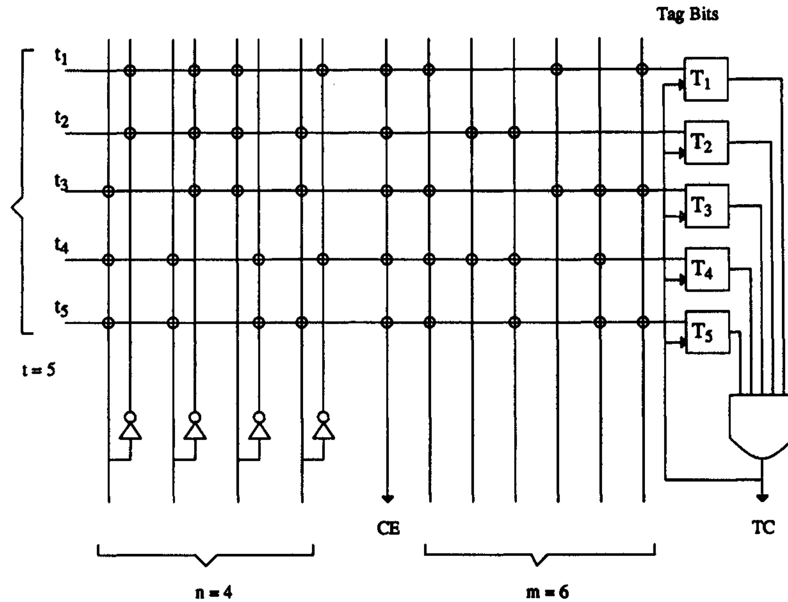


FIGURE 2 CTC for Example of Table I.

to keep track of the test vectors which have been HIT. The tag bits are initialized to logic 0 at the beginning of concurrent testing. Whenever a product line is activated during normal operation of the CUT, the corresponding tag bit is set to logic 1. Testing is said to be complete when all the tag bits in the PLA are at logic 1. The signal TC, logical AND of all the tag bits, marks the completion of the test. This signal can be used to reset all the tag bits to 0, thus initializing them for the next testing cycle. The CE signal is generated by ORing all the product lines of the PLA.

We must comment that the above method of implementation is but one way to implement CTC. Clearly, if the number of test vectors are large then the use of PLA or ROM may not be a practical alternative in terms of area overhead. Other alternatives such as use of an AND plane for storing the test vectors while using the duplicated CUT in the place of OR plane would need to be investigated. None the less the method suggested in Figure 2 can be used for circuits which are realized as iterative logic. Most such circuits can be tested by small and constant number of test vectors [9, 10].

The test hardware, which consists of the CTC and the comparator, can be shared, in time, by s identical CUTs (CUT#1, CUT#2, . . . , CUT#s) in an environment where these CUTs are either identical subcircuits of a larger circuit as in the case of iterative logic approach or when many copies of a CUT are used in a logic system. Further, it is possible to test each CUT in a roving manner. The testing can proceed as follows. Initially the test hardware tests CUT#1 and when CUT#1 has been tested, the test hardware is reconfigured to test CUT#2. Thus the process of testing can continue until the complete subsystem has been tested. The test hardware can be relegated to test any one CUT at a time by using multiplexers as shown in Figure 3. In this figure,

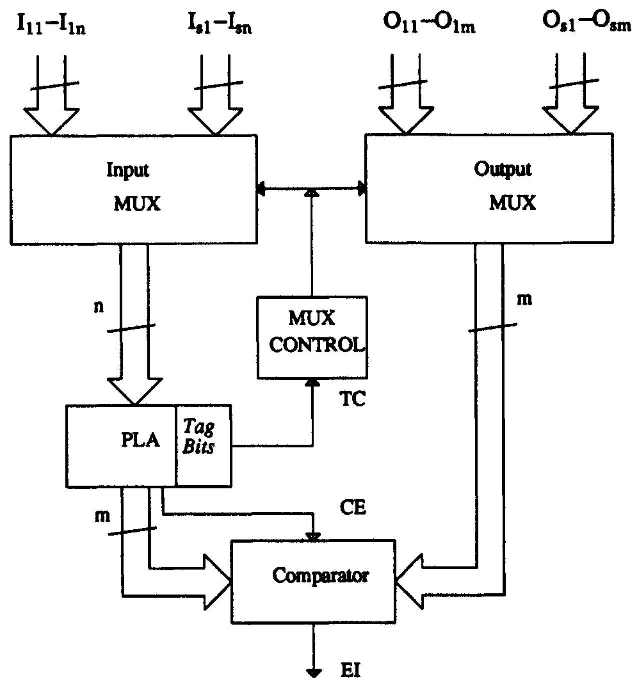


FIGURE 3 BICST Hardware for Testing s Indentical CUTs.

$I_{ij}(O_{ij})$ denotes the jth input (output) of CUT # i. The MUX control circuit controls the input and output multiplexors so as to allow the testing of the CUTs to proceed in a sequence. The TC signal from the CTC provides a feedback to the MUX control circuit so that the testing of the next CUT can be initiated. In the next section we shall discuss the implementation of the scheme shown in Figure 3 to design an ALU with built-in concurrent self-test (BICST) capability.

BICST DESIGN OF AN ALU

The scheme explained in Figure 3 was used to design an ALU with BICST capability, in CMOS technology. The ALU was made up of 4-bit 74181 ALU slices connected together in a ripple carry chain. The five main components of the CTC (units forming the on-line test circuitry) for a 12-bit ALU (made up of three 4-bit 74181 ALU slices) are briefly described below.

(1) **PLA:** The PLA is used to store the expected responses to test vectors for the ALU slice, as discussed in the previous section. Each ALU slice consists of 14 input-lines and 6 output-lines. The minimum number of test vectors [3] required to test the 4-bit 74181 ALU, available in the literature, is 12. Therefore, our PLA had 14 inputs, 7 outputs, and 12 product-lines. Note that one additional output from the PLA corresponds to the generation of the CE signal.

(2) **Tag bits:** Twelve latches were connected, one for each product line of the PLA, and the TC signal was generated by logically ANDing the outputs of these latches.

TABLE II
Area Overheads for Various ALU Sizes

ALU Size	Area Overhead (as % of total area)
4-bit	36%
8-bit	26%
12-bit	19%
16-bit	15%

(3) **MUX control:** The MUX control was implemented as a three stage ring-counter (one stage for each ALU slice) with preset capability. The ring-counter and the tag-bits are initialized at the beginning of the test.

(4) **Input and output MUX:** Each ALU slice has 5 common inputs (control signals) and 9 unique inputs (data signals). All 6 outputs of each ALU slice are unique. Therefore we had a (3 × 9) input-MUX and a (3 × 6) output-MUX. The input-MUX and the output-MUX were embedded at the ALU inputs and outputs respectively (Figure 4) to facilitate routing and to conserve chip area.

(5) **Comparator:** A 6-bit comparator was designed to compare the 6 outputs of the selected ALU with the 6 outputs obtained from the PLA when the CE signal is logic 1.

The floor plan and the relative areas of the various subsections of the system are given in Figure 4. The area of the BICST logic for the 12-bit ALU is 19% of total chip area. In this context the total chip area does not include pads but it includes the latches even though the latches are not tested in the present configuration. Details of the design are given in Sharma and Saluja [4]. The overall percent hardware overhead will decrease further if a 16 or higher bit ALU is designed using this scheme. The actual overhead for a 12-bit ALU and estimated overhead for 4, 8, and 16-bit ALUs is shown in Table II. It was con-

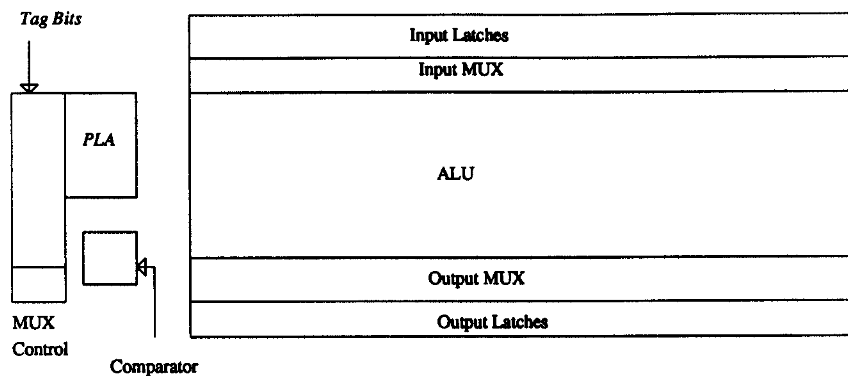


FIGURE 4 Floor Plan of a 12-bit ALU with BICST.

cluded from timing verification that the test hardware does not impose any time overhead on the normal operation of the ALU because the BICST hardware does not lie in the critical path.

The layout of the system was done using the layout editor MAGIC. The CAD tool MPLA was used to generate the PLA. Logic simulation and timing verification were done using ESIM and CRYSTAL respectively.

EVALUATION OF THE BICST TECHNIQUE

We have argued in the previous section that BICST is a conceptually sound method for concurrent testing and it is also a viable method for some classes of circuits such as an ALU realized as an iterative logic circuit. The question one can ask next is what is to be gained by concurrent testing in general and BICST as proposed here in particular? Clearly an answer to this question is required to justify the implementation of BICST. To answer this question we will first define certain parameters which can be used to evaluate a concurrent testing scheme. We will identify methods which can be used to compute these performance parameters. Following which we shall present the results of computation for the BICST as used for an ALU described earlier.

Below, we define three parameters which can be used as quantitative measures to determine the effectiveness of a concurrent testing scheme. These parameters are:

1. *Latency of Test Completion* (LTC): This is the time required to completely test the circuit while it is in normal mode of operation.
2. *Latency of Fault Detection* (LFD): This is the time required to detect a fault in the circuit under normal operation.
3. *Error Latency* (EL): This is the difference between the time when the fault is detected and the time when the fault manifests itself. (Shin and Lee [8])

These parameters can be determined only in probabilistic terms because the inputs to a circuit for its normal mode of operation cannot be known a priori. The parameters can be computed either analytically by using probability theory or through Monte Carlo simulation methods. In any event, we need to make some assumptions about the occurrence of inputs. For our study we shall make the following two assumptions:

A1. an input pattern occurs on every clock cycle, and

A2. the occurrence of any input pattern is statistically independent of the occurrence of any other input pattern.

First of these two assumptions is not limiting and the second assumption is realistic as the statistical dependence of the inputs cannot be known in advance. We shall now describe in detail the methods of computing the three parameters defined above and apply these methods to the example ALU.

Analytical Method to Compute Performance Parameters

For each of the parameters, we shall derive a general expression. An additional simplifying assumption will be made to keep the problem tractable. Following which the values of these parameters will be presented for our example subsystem, i.e. the ALU.

Computation of LTC

Consider a circuit employing BICST in which the CTC consists of t test vectors. Here we are interested in finding the probability $PTC(L)$ that within L normal input cycles, the CUT is completely tested. PTC(L) stands for *Probability of Test Completion* within L cycles of the normal system operation. This happens when a HIT corresponding to each of the test vectors of the CTC has occurred.

With the assumptions mentioned above, each normal input vector can be viewed as an independent Bernoulli trial. Each trial can have one of the $t + 1$ outcomes given below:

Outcome i: A HIT corresponding to the ith test vector of the CTC occurs ($i = 1, 2, \ldots, t$).

Outcome t + 1: None of the test vectors in the CTC are HIT.

Let p_i be the probability of outcome i ($i = 1, 2, \ldots, t$). Therefore the probability q of outcome $t + 1$ is given by:

$$q = 1 - \sum_{i=1}^{t} p_i \tag{1}$$

Now let us consider an experiment with L trials. Further, let $L = k_1 + k_2 + \cdots k_t + k_{t+1}$ where k_i denotes the number of occurrences of outcome i in L trials. If $P(L)$ is the probability that in L trials outcome i occurs exactly k_i times, then the value of $P(L)$ is given by Parzen [5]:

$$P(L) = \frac{L! p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t} q^{k_{t+1}}}{k_1! k_2! \cdots k_t! k_{t+1}!} \quad (2)$$

The CUT is completely tested when *at least* one HIT occurs for each test vector in the CTC. Therefore using (2), we can write the following expression for PTC(L):

$$PTC(L) = \sum_{k_1=1}^{L-i_1} \sum_{k_2=1}^{L-i_2} \cdots \sum_{k_t=1}^{L-i_t} P(L) \quad (3)$$

where

$i_r = K - k_r$; $k_i \geq 1$ for all $i = 1, \ldots, t$;

$$K = \sum_{j=1}^{t} k_j; \text{ and } L \geq K.$$

Latency of Test Completion with a desired probability α, denoted by $LTC(\alpha)$, can be calculated using expression (3). LTC(α) is the time after which the test completion probability is greater than or equal to α. If L_T is the smallest number of normal input cycles for which $PTC(L_T) \geq \alpha$, then $LTC(\alpha) = L_T \cdot \phi$, where ϕ is the time for one normal input cycle.

It can be extremely difficult to compute expression (3) for large L and for modest value of t. Also, it may be difficult, if not impossible, to know the values of p_is a priori. Therefore, we need to make an additional simplifying but realistic assumption. The additional assumption and its justification for the case of example ALU is given below:

A3. All input patterns are equally likely to occur.

At first glance this appears to be an unrealistic and very restrictive assumption. Let us consider the case of the ALU to justify this assumption. It is often argued that the numbers most likely to occur as inputs to the ALU during its normal operation will be small positive or negative numbers. If such an information is known a priori, we suggest that we choose a set of tests which is biased accordingly. If one is using 2's complement system, this statement is equivalent to having test vectors with larger proportions of 1s (0s) as opposed to equal number of 1s and 0s. Note that this would need to be done only for the most significant bit end of the ALU. The impact of the above assumption on the least significant bit end of the ALU will be minimal. Thus a choice of the appropriate test vectors will increase the probability of occurrence of certain test vectors above the value obtained with the assumption A3 while the probability of occurrence of other test vectors will be smaller. Hence, as a result of assumption A3 the overall outcome will be close to the actual value.

We must also comment on the limitation of the assumption A3 and possible ways to get around it. It is possible that in certain circuits some of the test vectors derived for off-line testing may never occur during normal operation of the circuit. This can happen due to the use of don't care conditions in the design of a circuit. For such circuits, (1) the test vectors should be carefully chosen so that vectors which may not occur as normal inputs are never used as test vectors, or (2) time out indicator, similar to the one to be discussed under Applications of the BICST Technique in this paper, can be used to determine the absence of test vector. Time out indicator can also be used to select an appropriate test set for the circuit. On the other hand, it can potentially be used to inject test vectors for the express purposes of testing a circuit at certain instances and thus causing reduced test latency.

In any case, in what follows we shall assume that the assumption A3 applies for each analysis. Thus for a CUT that has a total of N normal input vectors, we can assume:

$$p_1 = p_2 = \cdots = p_t = p = \frac{1}{N} \text{ and } q = 1 - \frac{t}{N}$$

Therefore the expression in (3) can be written as:

$$PTC(L) = \sum_{k_1=1}^{L-i_1} \sum_{k_2=1}^{L-i_2} \cdots \sum_{k_t=1}^{L-i_t} \times \frac{L! p^K q^{L-K}}{k_1! k_2! \cdots k_t! (L-K)!} \quad (4)$$

Even this expression is not easy to compute, hence we suggest to use the following method based on a rule of thumb to compute its approximate value. We observe that the probability of a HIT at the beginning of testing is t/N and it reduces after each HIT and eventually becomes $1/N$ when only one test vector remains to be HIT for test completion. In general, we denote $p(i) = i/N$ to be the probability of a HIT when i test vectors remain to be HIT. Then the cor-

responding mean number of cycles, $L(i)$ required for a HIT (when i test vectors remain to be HIT) is $1/p(i)$ or N/i (see Parzen [5]). Thus the *rule of thumb* number of cycles, L_{rot}, is given by expression (5) below.

$$L_{rot} = \sum_{i=1}^{t} L(i) = N\left(\sum_{i=1}^{t} \frac{1}{i}\right) \quad (5)$$

The computed values of L_{rot} for our ALU are given in Table III. These are found to be in close agreement with the simulation results (discussed later in this section).

Computation of LFD

The time required to detect a fault f depends on the number of test vectors in the CTC which can detect f. Let $t_D(f)$ denote the set of vectors in the CTC which detect fault f and let $|t_D(f)|$ be the cardinality of the set $t_D(f)$. The set $t_D(f)$ and its cardinality $|t_D(f)|$ can be determined at the time of coverage evaluation by fault simulation. The fault f will be detected if a HIT occurs corresponding to any element of $t_D(f)$. With the assumption that all input vectors are equally likely to occur, the probability that a vector from $t_D(f)$ occurs as a normal input on any cycle, denoted by p_D, is given by $p_D = |t_D(f)|/N$. Now the probability, $PD(f, L)$, that f is detected within L normal cycles is the same as the probability that at least one HIT for an element of $t_D(f)$ occurs within L cycles. This is given by Saluja et al. [2]:

$$PD(f, L) = \sum_{k=1}^{L} \binom{L}{k} p_D^k q_D^{L-k}$$

or

$$PD(f, L) = 1 - q_D^L \quad (6)$$

where

$$q_D = 1 - p_D$$

TABLE III
Values of LTC (in cycles)

ALU	Simulated Average	Simulated Worst	Simulated Best	Computed L_{rot}
4-Bit	49148	69560	25699	50843
8-Bit	85245	125940	63935	101686
12-Bit	156981	254033	87737	152529
16-Bit	211918	298152	149735	203372

The latency of detection of the fault f, denoted by $LFD(f, \alpha)$, can be calculated using $PD(f, L)$ obtained from expression (6), in a similar manner as $LTC(\alpha)$ was calculated using $PTC(L_T)$ in the section on Computation of LTC. By determining the value of $PD(f, L)$ for every fault the worst case value of this parameter can be determined. Alternatively, the average value of this parameter may be of more interest and can be determined as follows. If $F = \{f_i\}$ is the set of all faults of interest and the total number of faults in F is $|F|$ then the average number of test vectors in the CTC to detect any of these faults, e_{av}, is given by:

$$e_{av} = \frac{\sum_{i=1}^{|F|} |t_D(f_i)|}{|F|} \quad (7)$$

Therefore, the probability that a random fault from the set F will be detected within L normal cycles of its occurrence, denoted by $PD_{av}(L)$, is given by:

$$PD_{av}(L) = 1 - q_{Dav}^L \quad (8)$$

where

$$q_{Dav} = 1 - e_{av}/N \quad (9)$$

The latency of detection of a random fault, denoted by $LFD_{av}(f, \alpha)$, can now be calculated as before from $PD_{av}(L)$.

The value of e_{av} for our ALU is 3. We computed this value using a fault simulator and for the same set of test vectors which we used to design the CTC for the ALU. The fault set consisted of all single stuck-at-faults. For the 4-bit ALU, the computed value of LFD was found to be 12574 cycles for $\alpha = 0.9$ for a fault with $|t_D(f)| = 3$. For a larger ALU realized as iterative copies of the 4-bit ALUs, computation of LFD must take into account the test strategy. If each ALU is tested independently then the value of the LFD is same as shown above. On the other hand if each ALU is tested serially by sharing the CTC then much larger value of α is required. For a 12-bit ALU and for values of e_{av} and $|t_D(f)|$ given earlier the computed value of LFD for a fault will be $3 \times 12574 = 37722$ cycles for $\alpha = (0.9)^3 = 0.73$. For a system with 10MHz clock this is 3.77 msec.

Computation of EL

Let $EL(f, \alpha)$ be the time at which a fault f is detected, after its manifestation, with a probability α.

Then the error latency, $EL(f, \alpha)$, for the fault f is given by the expression:

$$EL(f, \alpha) = LFD(f, \alpha) - LFM(f, 1 - \alpha) \quad (10)$$

In the above expression $LFM(f, 1 - \alpha)$ is the latency of manifestation for the fault f with a probability $1 - \alpha$ or less. The evaluation of $LFD(f, \alpha)$ was done in the foregoing discussion. We shall now determine an expression for the evaluation of $LFM(f, \alpha)$.

Latency of manifestation of a fault f depends upon the number of input vectors, $m(f)$, which result in erroneous outputs in the presence of fault f. The probability that f manifests on any normal cycle, p_M, is given by $p_M = m(f)/N$. Therefore using similar arguments as in the derivation of (6) the probability that fault f manifests itself within L normal input cycles of its occurrence, $PM(f, L)$ can be written as:

$$PM(f, L) = 1 - q_M^L \quad (11)$$

Thus $LFM(f, 1 - \alpha) = \phi \cdot L_M$ where L_M is the smallest number of cycles for which $PM(f, L_M) \geq 1 - \alpha$.

The average error latency for the fault f can now be calculated by using (10). The error latency for the CUT, with a probability α on the occurrence of any fault at random, denoted by $EL_{av}(\alpha)$, is given by

$$EL_{av}(\alpha) = LFD_{av}(\alpha) - LFM_{av}(1 - \alpha) \quad (12)$$

$LFM_{av}(1 - \alpha)$ can be obtained by using q_{Mav} instead of q_M in (11) and q_{Mav} can be computed from the following expression:

$$q_{Mav} = 1 - \frac{\sum_{i=1}^{|F|} m(f_i)}{|F|} \quad (13)$$

The calculated values of LFD, LFM, and EL for the 4-bit ALU using BICST are given in Table IV for $|t_D(f)| = 3$ and for different values of $m(f)$. The values of EL for different values of $m(f)$ and larger size ALUs can be computed in the same manner as shown in the previous section.

TABLE IV
Evaluation of LFD, LFM, and EL

$m(f)$	Probability α	LFD(α) (cycles)	LFM($1 - \alpha$) (cycles)	EL (cycles)
6	0.5	3785	1892	1893
6	0.7	6575	973	5602
6	0.9	12574	287	12287
12	0.5	3785	946	2839
12	0.7	6575	486	6089
12	0.9	12574	143	12431
24	0.5	3785	473	3312
24	0.7	6575	243	6332
24	0.9	12574	72	12502

Simulation Method to Compute Performance Measures

A sequence of pseudo-random numbers satisfying the assumptions stated earlier in this section can be used as normal input vectors to the CUT for simulation purposes. The performance measures can then be computed by averaging over several simulation runs.

Such simulations were carried out to compute the LTC, LFD, and EL for our ALU. The results for LTC over 10 simulation runs are included in Table III where these values are also compared with the computed values using analytical method. A close agreement between calculated and simulated values was also observed for LFD and EL.

We shall now explore what is accomplished by the BICST scheme in terms of fault detection and the possible applications for this technique.

APPLICATIONS OF THE BICST TECHNIQUE

It is easy to see that the BICST technique helps in the detection of permanent faults while the circuit is in normal mode of operation. What additional benefits does BICST offer? In this section we argue that BICST can be used for system diagnostics and for reducing maintenance requirements. Further, BICST can also be used for detection of intermittent and transient faults. These issues are discussed in greater detail in the following subsections.

Detection and Diagnosis of Intermittent Faults

Intermittent faults are known to account for a large percentage of all system failures [6, 7]. These faults have always been a source of menace because they are subtle in nature and extremely difficult to detect and diagnose. Repeated application of test vectors during off-line testing has been suggested as a partial solution to this problem [6]. This approach is limited by the exorbitant test time when used in off-line testing environment. However, this approach can be effectively used to detect and diagnose intermittent faults during on-line testing using the BICST technique.

If a test vector t_i for a particular intermittent fault f_i appears a sufficiently large number of times, as a normal input vector to the circuit and the detection circuitry (comparator in the BICST technique) does not indicate any error, then we have a fair amount of confidence that the circuit is free of the intermittent fault f_i. Now the question we need to address is how long must we wait before we can be assured with a reasonable probability that the intermittent fault is not present?

A mathematical model was proposed in Kamal and Page [6] for the detection of intermittent faults by repeated application of test vectors during off-line testing. The same model and mathematical results can be used to estimate the number of applications of a test vector to reach a certain confidence level about the absence of intermittent fault. According to this model, the minimum number of times, $l_{\min}$, that a test vector must be applied to detect an intermittent fault (with a specified confidence) is given by the following expression [6]:

$$l_{\min} > \frac{log(\mu/\lambda)}{log(1 - g)} \tag{14}$$

where

μ = the probability that the fault goes undetected,
λ = fp/fn, fp is the probability that the fault is present (assumed to be known to start with) and $fn = 1 - fp$, and
g = the probability that the intermittent fault is active.

Note that λ and g are constants and assumed to be known a priori. Thus during on-line testing using BICST when a test vector for an intermittent fault occurs $l_{\min}$ times (and the detection circuitry does not indicate any error), then we are assured with a desired degree of confidence that the circuit is free of the intermittent fault. The probability that a particular input (test vector) occurs $l_{\min}$ times as a normal input within L normal input cycles is given by the expression:

$$P(l_{\min}, L) = 1 - \sum_{k=0}^{l_{\min}} \binom{L}{k} p^k q^{L-k} \tag{15}$$

Where p is the probability that the test vector occurs as normal input on any input cycle and $q = 1 - p$.

We shall illustrate these calculations with an example. Consider an intermittent fault for which $\lambda = 10^{-4}$, and $g = 0.0006$. Also, if we select $\mu = 10^{-8}$ then $l_{\min} = \frac{log(10^{-8}/10^{-4})}{log(1 - 0.0006)} = 15346$. If we further assume that only one test vector from the given test set is capable of detecting the intermittent fault under consideration, the time required for that test vector to appear 15346 times as normal input is obtained using the expression in (15). For our 4-bit ALU, we obtain $P(l_{\min}, L) = 0.99$ for $L = 275424870$. In other words, we have a 99% probability that the required test vector would appear 15346 times within 275424870 normal input cycles. If the ALU operates on a 10 MHz clock, then this would require 27.54 seconds. Computation of this value can also be carried out for larger ALUs as in the section on Computation of LFD knowing if the ALU slices are tested in parallel or serially by sharing logic.

An *Intermittent Test Complete* (ITC) signal can be generated in a similar manner the test complete (TC) signal was generated for detection of permanent faults. For each test vector in the CTC, instead of a single tag bit, a counter can be associated which counts to $l_{\min}$ for that test vector (the counter is reset by the ITC signal). Whenever a HIT occurs, the corresponding counter will be incremented. The ITC signal is generated when all the counters have counted to their respective values of $l_{\min}$. A more cost-efficient way to generate ITC would be to have a single counter which counts to the maximum value of $l_{\min}$ for the CUT and the TC signal can be used to increment this counter.

It must be noted that although the analysis in Kamal and Page [6] was done for the class of intermittent faults pertinent to off-line testing environment, the on-line testing method proposed here offers a method of detecting a wide range of intermittent faults including those which are related to system load [7]. Thus our method is capable of detecting faults related to system load conditions, the faults which will otherwise remain undetected by off-line testing methods.

Detection of Transient Faults

Transient faults occur during normal system operation due to several reasons like environmental conditions, stress or improper operating conditions. Transient faults are different from intermittent faults in that they do not recur. On-line testing can be used to detect the transient faults when they occur, and thus avoid the effects of erroneous circuit and system behavior. The coverage of transient faults by the online test techniques can be calculated as follows:

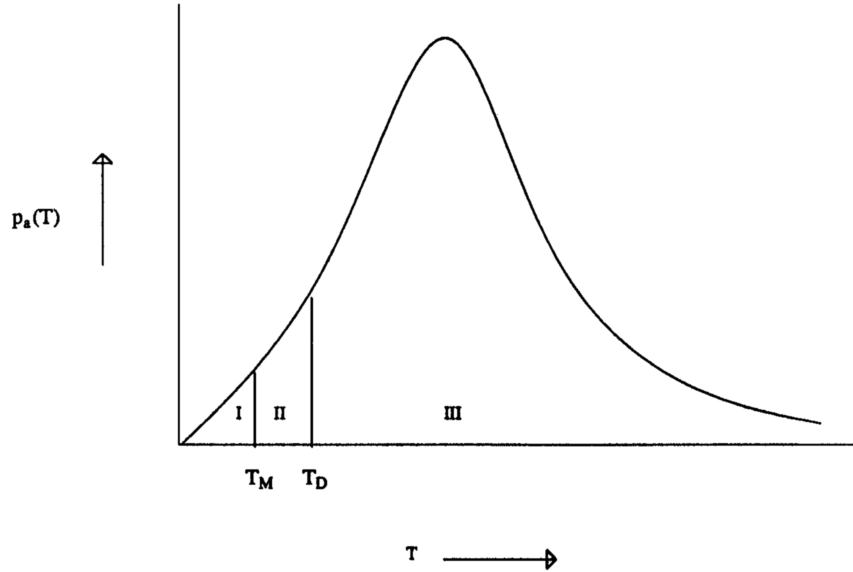


FIGURE 5 Transient Fault Coverage.

Transient faults have an active duration T_a, where T_a is a random variable with a probability density function $p_a(T)$. A fault is detected if its active duration is greater than its LFD. Note that LFD is computed by assuming a fault to be permanent. To determine the coverage of transient faults we carry out the following analysis by assuming that the faults are permanent. We then compute the detection probability of a fault at time T_D. If the detection probability at time T_D is α then the coverage of transient faults, $C(\alpha)$ is given by the expression

$$C(\alpha) = \int_{T_D}^{\infty} p_a(T)dT \qquad (16)$$

If the active duration of a transient fault is less than the time for the manifestation of the fault, T_M, then the fault would not produce any errors and would thus be harmless to the circuit operation (Figure 5). Thus, faults which have active duration longer than T_D are detected by our method (with probability α) and faults with active duration shorter than T_M are harmless. Therefore, we can consider the circuit employing BICST to be *error secure* during the interval 0 to T_M and T_D to ∞. The *error-secure coverage* (with a probability α), $ESC(\alpha)$, of transient faults is given by the expression:

$$ESC(\alpha) = \int_{T_D}^{\infty} p_A(T)dT + \int_{0}^{T_M} p_a(T)dT \qquad (17)$$

For a hypothetical curve of $p_a(T)$ shown in Figure 5, $C(\alpha)$ is the area under region III, and $ESC(\alpha)$ is the sum of areas under regions I and III.

We shall now illustrate the evaluation of $C(\alpha)$ for our 4-bit ALU. Let us assume for simplicity that the transient faults are uniformly distributed within the range of 0 to 20 msec [7]. If we assume that the ALU operates on a 10 MHz clock, then the values of $C(\alpha)$ are given in Table V for $|t_D(f)| = 3$. Once again $C(\alpha)$ and ESC(α) can also be computed for the larger size ALUs in a straightforward manner. The table

TABLE V
Transient Fault Coverage C(α) for 4-bit ALU

$m(f)$	α	t_D (msec)	t_M (msec)	$C(\alpha)$	$ESC(\alpha)$
6	0.5	0.38	0.19	98.1%	99.1%
6	0.7	0.66	0.09	96.7%	97.1%
6	0.9	1.26	0.03	93.7%	93.8%
24	0.5	0.38	0.05	98.1%	98.3%
24	0.7	0.66	0.02	96.7%	96.8%
24	0.9	1.26	0.01	93.7%	93.8%

exhibits a reasonably good coverage of transient faults by the BICST technique for the transient faults having the given distribution.

System Diagnostics

The TC and the ITC signals from various CTCs in a system can help in system diagnostics in the following manner. The TC and the ITC signals from the CTC can be used to place a time-stamp for the CUT. A record of each *Current Time-Stamp* (CTS) and *Previous Time-Stamp* (PTS) can be maintained. Now consider a situation where the system in which a CUT i, which is embedded in another circuit, is determined to have failed at some time T_{Fail}. As far as CUT i is concerned, we are sure that the only time during which a fault could have occurred is the time between PTS and T_{Fail}. The difference, T_{Fail} − PTS, is therefore a measure of the probability that the failing unit in the system is CUT i. This is shown in Figure 6. Thus from the diagnostics viewpoint, when many embedded CUTs are present in the system, the most probable CUT which may have failed is the one for which T_{Fail} − PTS is maximum.

Reduced Maintenance

Periodic off-line testing of circuits must be done to flush out latent faults, which could otherwise lead to system failure. In the absence of concurrent testing, the interval between maintenance schedules for periodic testing is set at the time *TMS* after which the probability of presence of latent faults reaches a certain threshold. (TMS stands for *T*ime for the next *M*aintenance *S*chedule.) With the concurrent testing technique proposed in this paper, the number of maintenance schedules can be reduced. This is achieved as follows.

We denote TC(i) to be the time at which the ith TC signal occurs from the CTC. Also, TMS(i) represents the TMS after the occurrence of TC(i). At the beginning of concurrent testing, TMS(0) (= TMS) is the original TMS (see Figure 7). If TC(1) occurs before TMS(0), then maintenance can be postponed until TMS(1) (= TMS(0) + TC(1)). Thereafter, if TC(2) occurs before TMS(1), then maintenance can be postponed until TMS(2) (= TMS(1) + TC(2)). The generic expression for TMS at any given time during normal operation is given by:

$$TMS(i) = TMS(i-1) + TC(i) \qquad (18)$$

It should be evident from the above discussion that no maintenance schedule would be required so long as LTC is less than TMS. This results in improved system availability.

CONCLUSIONS AND COMMENTS

We have proposed a technique for on-line testing of digital circuits in this paper. We have also discussed a design of an ALU using BICST, the on-line test technique proposed by us. The ALU was designed in the CMOS technology. The results indicate a modest area overhead for implementation of BICST. We

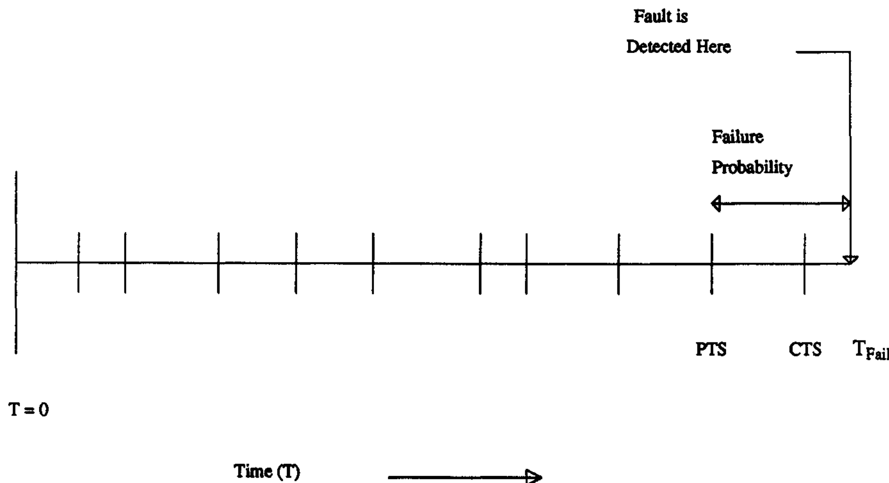


FIGURE 6 System Diagnostics.

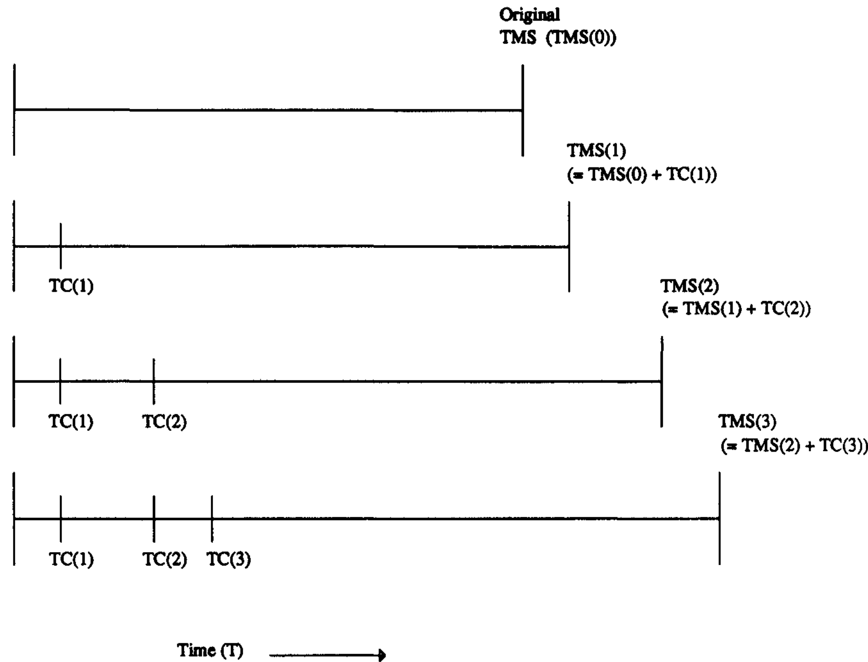


FIGURE 7 Reduced Maintenance.

have defined a set of parameters for the performance evaluation of on-line test techniques. These parameters are general enough and can be used to evaluate not only on-line test techniques but can also be used to evaluate on-line checking techniques. Following the definition of these parameters, we developed analytical and simulation methods to compute these parameters for the BICST technique proposed in this paper. The results of these computations show that BICST can be a viable alternative for improving reliability and availability. We have also argued that BICST technique can be used for detection and diagnosis of transient and intermittent faults.

A comparison between circuits employing BICST and self-checking circuits is interesting. Unlike BICST, the self-checking circuits provide 100% protection against errors due to transient faults (fault-secure property). The amount of protection against transient faults with the BICST technique (Table V) depends upon the distribution of these faults and it can be increased to an arbitrarily high value by increasing the number of test vectors in the CTC at the expense of hardware. In the extreme case when the CTC contains all input vectors as test vectors, then we obtain the goal of 100% protection against transient faults. Note that this situation is the same as duplication of the CUT.

Although the self-checking circuits can lead to the detection of intermittent faults, there is no means of determining whether the circuit is free of intermittent faults. The circuits with BICST on the other hand can help in diagnosing the intermittent faults with the help of the test completion signal.

The fault-secure property of self-checking circuits ensures that any error due to a fault within the prescribed *fault-model* for the self-checking circuit will be detected. The fault-secure property may be invalidated on the occurrence of a fault which violates the fault model. The fault model for designing self-checking circuits is generally the single stuck-at fault model. This means that the self-checking circuits may lose their fault-secure property in the presence of multiple faults. At any given time a self-checking circuit is said to be *model-secure* if the fault(s) in the circuit (if any) are confined to the fault model. What is therefore desirable is to have some means to ensure that a self-checking circuit remains model-secure at all times. It is exactly this job that is done by the BICST technique. The occurrence of the TC signal within some predetermined time can be used to decide that the circuit is model-secure.

Acknowledgments

The authors are grateful to Prof. C.R. Kime, Dr. Vishwani Agrawal and the anonymous referees for their suggestions to improve the presentation of the paper. This work was supported in part by the University of Wisconsin Graduate Research Committee, National Science Foundation under grants MIP 8509194 and MIP 911886.

Glossary

α	Confidence level, expressed in %
ϕ	Cycle time for normal inputs
BICST	Built-In Concurrent Self-Test
$C(\alpha)$	Coverage of transient faults with a probability α
CE	Comparator Enable
CTC	Concurrent Test Circuit
CTS	Current Time-Stamp
e_{av}	Average number of test vectors present in the CTC to detect any fault in the CUT
EI	Error Indicator
EL	Error Latency
$EL(f, \alpha)$	Error latency of a fault f with a probability α
$EL_{av}(\alpha)$	Error latency for the CUT, with a probability α on the occurrence of any fault at random
$ESC(\alpha)$	Error secure coverage of transient faults with a probability α
f	A fault on the CUT
F	($= \{f_i\}$) Set of all faults of interest in the CUT
$\|F\|$	Total number of faults present in the CUT
HIT	Condition when the normal input vector to the CUT matches a test vector of the CTC
ITC	Intermittent Test Complete
$l_{\min}$	Minimum number of applications of a test vector required to reach a desired confidence level about the absence of the intermittent fault under consideration
L_{rot}	Number of cycles (obtained by using the *rule of thumb*) required for completely testing the CUT
LFD	Latency of Fault Detection
$LFD(f, \alpha)$	Latency of detection for fault f with a probability α
$LFD_{av}(f, \alpha)$	Latency of detection of a random fault with a probability α
$LFM_{av}(f, \alpha)$	Latency of manifestation of a random fault with a probability α
$LFM(f, \alpha)$	Latency of manifestation for a fault f with a probability α
LTC	Latency of Test Completion
$LTC(\alpha)$	Latency of test completion with a probability α
m	Number of output lines of the CUT
$m(f)$	The number of input vectors which result in erroneous outputs in the presence of the fault f
n	Number of input lines of the CUT
N	Total number of input vectors for the CUT ($=2^n$)
p	Probability that a particular test vector is HIT on any normal input cycle
$p_a(T)$	Probability density function of T_a
p_D	Probability that an element of $t_D(f)$ occurs as a normal input on any cycle
$P(l_{\min}, L)$	Probability that a particular test vector occurs $l_{\min}$ times as normal inputs within L normal input cycles
p_M	Probability that the fault f manifests itself on any normal input cycle
$PD_{av}(L)$	Probability that a fault at random will be detected within L normal cycles of its occurrence
PD(f, L)	Probability that the fault f is detected within L normal input cycles
PM(f, L)	Probability that the fault f manifests itself within L normal input cycles of its occurrence
PTC(L)	Probability that the CUT is completely tested within L normal input cycles
PTS	Previous Time-Stamp
q	Probability that none of the test vectors is HIT on any normal input cycle
q_D	Probability that no element of $t_D(f)$ occurs as a normal input on any cycle
q_{Dav}	Probability that on an average no element of $t_D(f)$ occurs as a normal input on any cycle
q_M	Probability that the fault f does not manifest itself on any normal input cycle
q_{Mav}	Probability that on an average the fault f does not manifest itself on any normal input cycle
T_a	Active duration of a transient fault
T_D	Time at which a fault is detected
$t_D(f)$	Set of test vectors in the CTC which detect the fault f
$\|t_D(f)\|$	The number of elements in the set $t_D(f)$
T_M	Time at which a fault is manifested
TC	Test Complete
T_{Fail}	Time at which the system in which the CUT is embedded is detected to have failed
TMS	Time between Maintenance Schedules

References

[1] T.W. Williams and K.P. Parker, "Design for Testability—A Survey," *Proc. IEEE,* 71(1), 98–112, January 1983.

[2] K.K. Saluja, R. Sharma, and C.R. Kime, "A Concurrent Testing Technique for Digital Circuits," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems,* 7(12), 1250–1260, December 1988.

[3] S.B. Akers and B. Krishnamurthy, "Test Counting: An Analysis Tool for VLSI Testing," Tech. Report CR-86-55, Computer Research Laboratory, Tektronix Laboratories, November 1986.

[4] R. Sharma and K.K. Saluja, "Design, Analysis and Applications of an On-Line BIST Technique," Technical Report ECE-88-10, Department of Electrical and Computer Engineering, University of Wisconsin—Madison, June 1988.

[5] E. Parzen, *Modern Probability Theory and Its Applications.* New York: John Wiley, 1960.

[6] S. Kamal and C.V. Page, "Intermittent Faults: A Model and Detection Procedure," *IEEE Transactions on Computers,* C-23(7), 713–719, July 1974.

[7] R.K. Iyer and D.J. Rossetti, "A Measurement-Based Model for Workload Dependence of CPU Errors," *IEEE Transactions on Computers,* C-35(6), 511–519, June 1986.

[8] K.G. Shin and Y. Lee, "Measurement and Application of Fault Latency," *IEEE Transactions on Computers,* C-35(4), 370–375, April 1986.

[9] H. Elhuni, A. Vergis, and L. Kinney, "C-Testability of Two-Dimensional Iterative Arrays," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems,* 5, 573–581, July 1986.

[10] W.K. Huang and F. Lombardi, "On an Improved Design Approach for C-Testable Orthogonal Iterative Arrays," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems,* 7(5), 609–615, May 1988.

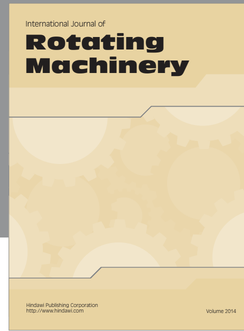
Biographies

RAJIV SHARMA received his B.E. degree in Electronics from Punjab Engineering College in India. He obtained his masters degree in Electrical Engineering from Michigan Technological University, and worked as a Research Assistant in the area of design for testability at the University of Wisconsin—Madison. Currently, he is working as a Software Engineer responsible for design and development of test products at Cadence Design Systems.

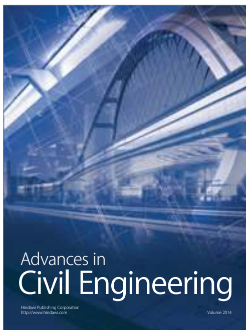
KEWAL K. SALUJA received his B.E. degree in Electrical Engineering from the University of Roorkee, India, and M.S. and Ph.D. degrees from the University of Iowa.

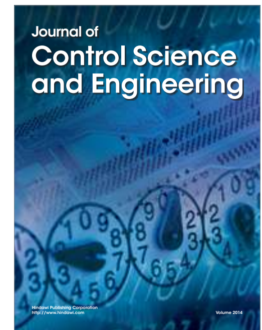
He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Wisconsin—Madison where he teaches logic design, computer architecture, microprocessor based systems, and VLSI design and testing. Previously he was at the University of Newcastle, Australia. He has also held visiting and consulting positions at a number of institutions including the University of Southern California, the University of Iowa, the State University of New York, and Hiroshima University. His research interests include design for testability, fault-tolerant computing, VLSI design and computer architecture and he has authored or co-authored over 100 research papers in these areas.

Journal of
Engineering

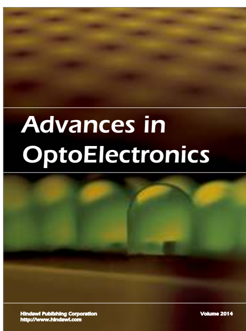
The Scientific
World Journal

International Journal of
Rotating
Machinery

Journal of
Sensors

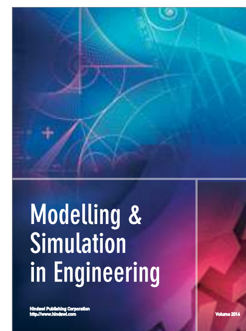
International Journal of
Distributed
Sensor Networks

Advances in
Civil Engineering

Hindawi
Submit your manuscripts at
http://www.hindawi.com

Journal of
Control Science
and Engineering

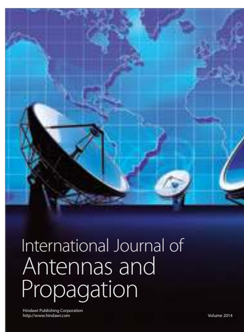
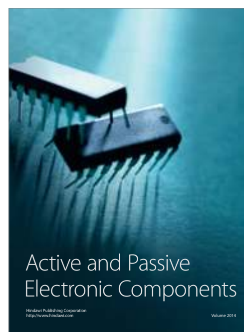
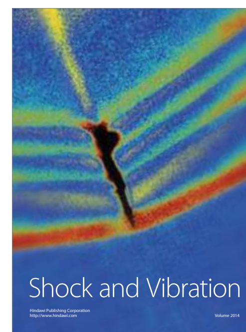
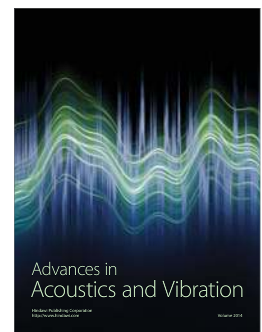
Journal of
Robotics

Journal of
Electrical and Computer
Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and
Observation

Modelling &
Simulation
in Engineering

International Journal of
Aerospace
Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and
Propagation

Active and Passive
Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration