A Concurrent Testing Technique for Digital Circuits

KEWAL K. SALUJA, MEMBER, IEEE, RAJIV SHARMA, AND CHARLES R. KIME, SENIOR MEMBER, IEEE

Abstract—**In this paper, we present a method of testing digital circuits during normal operation. The resources used to perform on-line testing are those which are inserted to alleviate the off-line testing problem. The off-line testing resources are modified such that during system operation they can also observe the normal inputs and outputs of a combinational circuit under test. The normal inputs to the circuit under test are compared with test vectors in its test set. When a normal input matches a test vector, the circuit output for such an input is typically compressed into a developing signature. When all of the test vectors in the test set have appeared as normal inputs, the signature is read and verified.**

With this method, the length of time required for all of the test vectors to appear, possibly in some order, among the normal inputs to the circuit under test is of considerable importance. We refer to this as the test latency and give analytical methods for its computation with verification by simulation. We also describe a hardware structure for implementing the concurrent test method and identify a number of approaches for reducing test latency.

Index Terms—**Concurrent testing, test latency, built-in-self-test, VLSI testing, testable design.**

I. Introduction

WITH THE advent of VLSI, the complexity of digital circuits has been increasing at an exponential rate. Unfortunately, increased circuit complexity also complicates the testing problem. To ease testing, two new disciplines have emerged, namely, *design for testability* (DFT) and *built-in self-test* (BIST) [1]. Both these disciplines require adding extra logic to alleviate the testing problem. In DFT environment, the extra logic is active during the test mode (off-line) operation of a logic circuit and such logic is idle during normal operation of the circuit. BIST techniques, on the other hand, can be used either for on-line or for off-line testing.

Structures used for on-line and off-line BIST are different. This is so because on-line BIST uses normally occurring data as inputs and employs redundancy techniques such as information redundancy, time redundancy, and hardware redundancy to do concurrent checking. Off-line BIST, on the other hand, uses stored and/or generated test vectors as inputs and employs compression and comparison techniques for off-line testing. As a result, typical structures used by on-line BIST are checkers and duplicated or triplicated hardware with comparators [1], whereas the structures employed for off-line BIST include the *linear feedback shift register* (LFSR), *built-in logic block observer* (BILBO) [2], or binary counter [3], often with such features as scan paths [1]. During normal operation, the on-line BIST hardware is active, while only a portion of the off-line BIST hardware is active and the remainder is idle. Furthermore, even in the presence of on-line BIST techniques, it is often essential for digital circuits to have either DFT or off-line BIST features for production testing and field diagnostics.

In this paper we propose ways of effectively utilizing the logic added for off-line BIST to perform testing of the circuit concurrently with its normal operation with little additional impact on the circuit performance. Concurrent testing as proposed here is achieved by observing the normal inputs to the circuit and the responses of the circuit to these normal inputs. This technique of concurrent testing is called *concurrent comparative built-in self-test* (C-BIST). The hardware structure used to implement C-BIST is known as a *Concurrent comparative built-in self-test unit* (CBU). A CBU is capable of generating test vectors and verifying responses during both off-line and on-line modes of operation of the circuit to be tested.

The use of this technique has several advantages relative to DFT and off-line BIST methods. First of all, it is possible in the production environment to run BIST tests concurrently with functional chip and system tests, thereby providing much more thorough testing in less time. Second, in the field environment, faults are detected fairly rapidly thereby reducing the potential corruption of data. Furthermore, added information is available on a continuing basis that is useful in locating faults and helps to reduce the number of diagnostic tests required. In addition, the technique is useful in conjunction with on-line BIST to detect multiple faults in the production environment and to provide guaranteed self-testing for self-checking in the field environment.

In Section II we describe some of the existing off-line BIST methods which are relevant to C-BIST. Section III describes the C-BIST concept. Methods of computing test latency for CBU's, an important measure for evaluating the C-BIST technique, are discussed in Section IV. The design of a hardware unit for concurrent testing is given in Section V. Finally, discussion and conclusions are given in Section VI.

Manuscript received February 15, 1988; revised July 12, 1988. This work was supported in part by the University of Wisconsin Graduate Research Committee and by the National Science Foundation under Grant MIP 8509194. The review of this paper was arranged by Associate Editor V. K. Agarwal.

This is an expanded version of the work originally presented at the ICCAD-87.

K. K. Saluja and C. R. Kime are with the Department of Electrical and Computer Engineering, University of Wisconsin at Madison, Madison, WI 53706.

R. Sharma was with the University of Wisconsin at Madison, Madison, WI 53706. He is now with Gateway Design Automation Corporation, Lowell, MA 01852.

IEEE Log Number 8824032.

II. Existing Off-Line BIST Techniques

The conventional techniques for off-line testing make use of test pattern generation and fault simulation algorithms to find a set of test vectors to detect the modeled faults in a circuit. These tests can be applied to the circuit either by an external tester or alternatively they can be stored on chip and applied during the test mode. The method of storing tests on chip and applying the test vectors during the test phase can be viewed as an off-line BIST technique. Various other off-line BIST techniques have been proposed in the literature, some of which are outlined below. The exhaustive testing [1] technique does not require a fault model and test generation. Also, it does not require that the test vectors be stored, because the test vectors can be generated by a *test generator* (TG) such as a counter or an LFSR with nonlinear circuitry added to include the all zero pattern. However, the method cannot be used for circuits with large numbers of inputs because the time for testing a circuit becomes large. To overcome this limitation, other techniques such as pseudoexhaustive testing [4], verification testing [5], and testing using random inputs [6] have been proposed.

To reduce the storage requirements for expected responses, the signature analyzer [7] has been proposed as a *response verifier* (RV). Output responses can be compressed by using, for example, LFSR's or *multiple input linear feedback shift registers* (MISR's) as RV's while the test vectors are being applied. The final compressed response (known as a signature) is compared with the expected response. Another technique, known as syndrome testing [3], tests the combinational circuit exhaustively while compressing the output. One of the features of this method, which signature methods lack, is that the compressed output is independent of the order in which the input test vectors are applied, although the compressor to compress the responses of a multi-output circuit can be rather complex. Further, the *circuit under test* (CUT) can be made to be syndrome testable, although at the expense of increasing the number of inputs to the circuit.

A structure which combines both the TG and the RV functions is the *built-in logic block observer* (BILBO) [2]. The BILBO and its variations can be used in different modes as an LFSR, an MISR or a part of a scan chain during the off-line test phase. This structure can be used in the pseudoexhaustive or exhaustive testing environment.

A general organization of hardware for an off-line BIST environment is shown in Fig. 1(a). Note that only the combinational logic is shown in the test path. If the circuit realizes a finite state machine then the feedback is tested in a separate test phase. *Test logic and normal logic* blocks contain the necessary hardware for the test as well as for the normal mode of operation. The normal logic parts of the test logic and normal logic blocks in Fig. 1(a) typically consist of latches. The test mode operation of the circuit is shown in Fig. 1(b). During normal mode, parts of the test generator, TG, and the response verifier, RV, are idle as shown in Fig. 1(c). Such a portion of the test

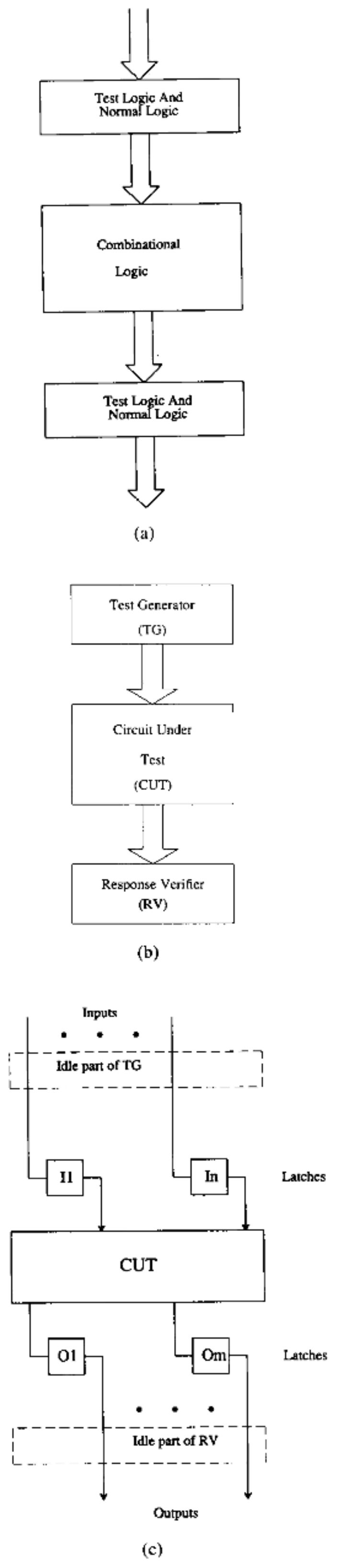


Fig. 1. A BIST organization. (a) Off-line BIST organization. (b) Test mode. (c) Normal mode.

logic is called *idle test logic*. Clearly, the idle test logic, which was introduced to alleviate the off-line testing problem, performs no useful function during the normal system operation. On the other hand, if the idle test logic is sufficient to act as a TG or RV, then it can potentially be used for concurrent testing. It is this concept we first consider at some length in the next section. Note that to impart this power to the idle test logic it may be necessary to increase the amount of test hardware. The details of the hardware design for concurrent testing will be discussed in a later section.

III. C-BIST Concept

To illustrate the C-BIST concept, let us consider a BIST scheme, in which the combinational part of a circuit with n inputs is tested exhaustively by using the circuit organization shown in Fig. 2. In this organization, we assume that the TG and the RV stay idle during the normal mode of operation of the circuit. We have chosen this representation for clarity of presentation. An actual organization may not employ *multiplexers*, or only parts of TG and RV may form idle test logic. We shall comment on these details in later sections.

We modify the hardware associated with the idle test logic as shown in Fig. 3. We have added an equality comparator to compare the normal inputs to the CUT to the outputs of the TG. The structures shown within dotted lines form the CBU. The two modes of testing proceed as follows.

Test Mode:

This is off-line testing mode. During this mode, the multiplexer is set such that the tests generated by the TG are applied to the CUT and the responses are compressed using the RV. After the application of all tests, contents of the RV are used to determine the status of the CUT. In this configuration of the circuit, the equality comparator plays no role at all.

Normal Mode:

This is the system operation mode and the testing of the CUT proceeds concurrently with the normal operation of the system. In order for the RV to obtain a result identical to that for the test mode, it must process only those responses that correspond to the test input vector set or test input sequence applied during the test mode. Thus the RV must be enabled only if a vector from a particular set or a vector at a particular point in a test sequence occurs. (The choice of a vector set versus a vector sequence depends on the type of RV employed.) The enabling of the RV is accomplished as follows. The TG produces, at its outputs, vectors from the test set or sequence for the CUT. During normal operation, the outputs of the TG are constantly compared, bit-by-bit, with the normal inputs to the CUT by the equality comparator. The particular output vector with which the normal inputs are compared is known as the *active test vector*. A signal HIT is generated if and only if the normal inputs to the CUT are the same as an active test vector of the TG. When a HIT is generated, the RV is enabled to compress/record the normal

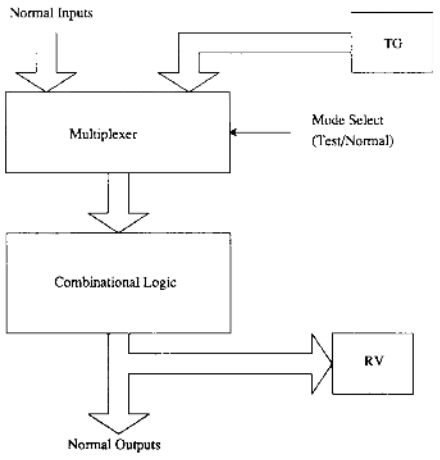


Fig. 2. An off-line BIST organization.

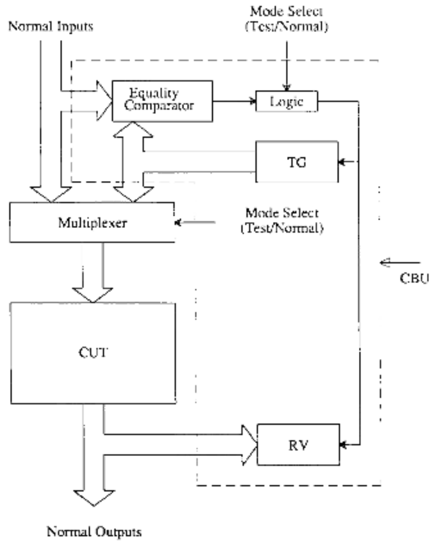


Fig. 3. Modified BIST organization.

outputs of the circuit. Also, the TG is advanced to the next state to produce the next vector only when a HIT occurs. It is possible to have multiple active test vectors; when the HIT signal is activated, the particular active test vector present is identified. When the TG has gone through the appropriate set of states, i.e., all test vectors in the set or sequence have been applied to the inputs of the CUT, then the contents of the RV are examined to determine the status of the CUT.

From the above conceptual explanation, it should be evident that the testing of the circuit can proceed concurrently with the normal operation of the circuit.

IV. Determination of Test Latency

An important evaluation measure for the performance of the CBU's is the time required for test completion. Let T be the set of test vectors to completely test the CUT. Clearly the fault-free status of the CUT cannot be verified until all the test vectors from T have been applied to the

CUT. For the CBU discussed in the previous section (Figure 3), the status of the circuit cannot be determined until the TG has gone through all the required states. The question we need to address is the time required for the TG to go through all the states necessary to completely test the circuit.

Definition 1: The test latency (TL) for a circuit is the time required for the TG to go through all states corresponding to the test vectors in the test set T.

Definition 2: Suppose that the TG is an exhaustive test pattern generator with n storage elements and it produces all possible 2^n combinations as the test set T. Then the *test latency of exhaustive testing (TLET)* for a circuit is the time required for the TG to go through all possible 2^n states.

Note that for exhaustive testing purposes the TG could be implemented as a counter or an LFSR with nonlinear circuitry added to include all-zero pattern. In such a case the state of the TG can be considered as the output of the TG. In what follows we will compute the values of TL and TLET. The following statement is the direct consequence of the above definitions.

In test mode, i.e., off-line testing, assuming that a test is applied every clock cycle,

$$\text{TL} = |T| \cdot T_c$$

$$\text{TLET} = 2^n \cdot T_c$$

where $|T|$ denotes the number of test vectors in the test set T, T_c is the clock period, and n is the number of inputs to the CUT.

To compute TL or TLET during the normal mode of operation, we make two assumptions. These are (1) all input patterns are equally likely to occur during the normal operation of the circuit and (2) the probability of occurrence of any one pattern is independent of the occurrence of any other pattern. We have resorted to these assumptions because in many practical circuits information about the occurrence of normal input patterns and their statistical dependence is not known *a priori*. These assumptions therefore provide a general method of computing TL for a circuit during normal operation. We must add, however, that if the statistics of normal input patterns are known, then it may be possible to design a test generator which would result in shorter test latency compared to that obtained by making the above assumptions. For example, if it is known *a priori* that the normal inputs occur in an ascending order as a count sequence, then TG realized as a counter will result in a short test latency. On the other hand, it may happen that some inputs may occur rather infrequently during normal system operation. In such a case the test latency may be longer. Problems associated with the situation where some inputs never occur during normal operation are discussed in Section VI.

In the rest of this section we shall assume that the TG is used for exhaustive testing; therefore, our discussion will be confined to the computation of TLET. Further, we will compute the value of TLET in terms of the number of clock cycles, assuming that one and only one input is applied to the CUT in any one clock cycle. It should be pointed out that very often it is not necessary to apply an exhaustive set of test patterns to test a circuit. A desired fault coverage can generally be obtained with a much shorter sequence of random test vectors. Therefore, this paper gives a rather pessimistic value of test latency for practical purposes. We observe that TLET in the normal mode of operation can only be determined in a "probabilistic sense" as defined below:

Definition 3: Concurrent test latency for exhaustive testing is the time it takes for the TG to go through all possible 2^n states with a probability α during the normal operation of the circuit. We shall denote it as $C(\alpha)$.

In the above definitions $C(\alpha)$ denotes the time required to test the CUT exhaustively during normal system operation. Thus α is the level of confidence we have that during the period $C(\alpha)$ the TG goes through 2^n states. Clearly it is desirable that α be close to 1. If $V_{C(\alpha)}$ denotes the actual number of states the TG has gone through (i.e., the actual number of tests applied to the CUT which were also perceived to be HITs by the TG) in time $C(\alpha)$ then

$$\lim_{\alpha \to 1} V_{C(\alpha)} \to 2^n.$$

4.1. Computation of Latency

Test latency can be computed either analytically by using probability theory or through Monte Carlo simulation. In the next two subsections we discuss these methods to compute test latency in greater detail.

4.1.1. Analytical Technique to Compute Test Latency: The TG is reset at the initiation of an on-line testing phase. Let L be the number of cycles, since the last reset, during which the TG has been in on-line test mode. As explained earlier, during L cycles, whenever a normal input matches the active test vector of the TG, a HIT is said to occur. After every HIT, the TG advances to the next state producing the next active test vector. On the other hand if a normal input does not match the active test vector of the TG, a MISS is said to occur and the TG does not change state after a MISS. Thus, every cycle can be viewed as an independent Bernoulli trial resulting in a HIT or a MISS. From the assumptions it is evident that the probability of a HIT, *denoted by* p, is $1/2^n$ and that of a MISS, *denoted by* q, is $1 - (1/2^n)$. Also, these probabilities are independent of the active test vector and hence the state of the TG during any cycle.

Thus the sample space for our experiment consists of L independent Bernoulli trials with $p = 1/2^n$ and $q = 1 - (1/2^n) = 1 - p$. With respect to our objective, the testing will be complete if in L such trials we have at least 2^n HIT's. Note, the fact that the active test vector of TG changes after a HIT does not affect the probability of HIT, although due to change in the active test vector, the definition of HIT changes. The probability of exactly k HIT's in L such trials is given by

$$\binom{L}{k} p^k q^{L-k} \tag{1}$$

where

$$\binom{L}{k} = \frac{L!}{k!(L-k)!}.$$

Clearly, the testing will not be complete if there are fewer than 2^n HIT's in L trials. Therefore the *probability of incomplete testing*, $P(TI)$, is the probability of fewer than 2^n HIT's in L trials. This probability is given below:

$$P(TI) = \sum_{k=0}^{k=2^n-1} \binom{L}{k} p^k q^{L-k}. \tag{2}$$

Therefore, the *probability of the completion of test*, $P(TC)$, is

$$P(TC) = 1 - P(TI). \tag{3}$$

(Note: To be exact, the expression for $P(TC)$ should be

$$P(TC) = 1 - P(TI), \quad \text{for } L \geq 2^n$$
$$= 0, \quad \text{for } L < 2^n.$$

However, we shall only be interested in the case when L is much larger than the number of tests to be applied.)

Clearly, it is quite laborious to compute the above expression even for small values of n because 2^n is very large. However, using the De Moiver–Laplace theorem [8, p. 239] the expression in (2), which obeys the binomial probability law, can be approximated as follows:

$$\begin{aligned} P(TI) &= \sum_{k=0}^{k=2^n-1} \binom{L}{k} p^k q^{L-k} \\ &\approx \Phi\left(\frac{2^n - 1 - Lp + (1/2)}{(Lpq)^{(1/2)}}\right) \\ &\quad - \Phi\left(\frac{0 - Lp - (1/2)}{(Lpq)^{(1/2)}}\right). \end{aligned} \tag{4}$$

In the above equation $\Phi(x)$ denotes the normal distribution of x. We must add that there exists a Poisson approximation for (2), [8] and one should be careful in choosing the right approximation. The approximation given in (4) is used because Lp is expected to be large. Clearly, to determine $C(\alpha)$, for given α we need to determine the smallest L such that

$$P(TC) \geq \alpha \tag{5}$$

or

$$P(TI) < 1 - \alpha. \tag{6}$$

The following example demonstrates the use of above equations for computing the value of the concurrent test latency for exhaustive testing.

Example: Computation of C(0.99) for a 10-Input Circuit:

Using (6) and (4) we need to determine the smallest L such that

$$\Phi\left(\frac{2^n - 1 - Lp + (1/2)}{(Lpq)^{(1/2)}}\right) - \Phi\left(\frac{-Lp - (1/2)}{(Lpq)^{(1/2)}}\right) < 0.01 \tag{7}$$

and *a fortiori* due to the fact that $\Phi(\cdot)$ is a positive function

$$\Phi\left(\frac{2^n - 1 - Lp + (1/2)}{(Lpq)^{(1/2)}}\right) < 0.01. \tag{8}$$

Using the normal distribution table we find that the above inequality is satisfied provided

$$\frac{2^n - 1 - Lp + (1/2)}{(Lpq)^{(1/2)}} < -2.33.$$

For a 10-input circuit $n = 10$, $p = 1/2^n = 1/1024$ and $q = 1023/1024$. Substituting these values in the above inequality and solving the resulting quadratic equation we find

$$\frac{L}{2^n} \geq 1100.7$$

or

$$L \geq 1127117.$$

For this value of L the value of the term dropped between (7) and (8) is negligible. Therefore, $L = 1127117$ is a very good approximation for $C(0.99)$ for this example. For a 10-MHz system clock, this results in $C(0.99) = 112.7$ ms.

$C(\alpha)$ expressed as the number of system clocks and time units for a 10-MHz system for various values of α and n is given in Table I.

4.1.2. Monte Carlo Simulation Technique to Compute Test Latency:

A sequence of pseudorandom numbers satisfying the two assumptions stated earlier in this section can be used as the normal input vectors for simulation purposes. The average test latency can then be computed by averaging over several simulation runs. Note that the average value of L so obtained corresponds to $\alpha = 0.5$.

Such simulations were performed for circuits with various number of inputs. The results of these simulations are shown in Table II. The value of L obtained from (4) for $\alpha = 0.5$ is approximately $(2^n - 0.5)2^n$. In this table a comparison is also made between the test latencies obtained using analytical expressions and using simulation results. It is evident from the table that the simulation results closely match the analytical results. A close look at Tables I and II reveals that only a small increase in L increases the confidence level α substantially. This point is also illustrated by the simulation experiments. The maximum and the minimum test latencies were within at

TABLE I
$C(\alpha)$ FOR DIFFERENT VALUES OF n AND α

n	C(0.90)		C(0.95)		C(0.99)	
	L	sec	L	sec	L	sec
10	1090891	0.11	1103330	0.11	1127044	0.11
12	17114557	1.71	17211892	1.72	17395982	1.74
14	271129177	27.11	271899147	27.19	273349519	27.33

TABLE II
AVERAGE TEST LATENCY

n	Average Test Latency (L)		Difference Between Analytical and Simulated Results
	Analytical	Simulated	
10	1048064	1048975	0.09%
12	16775168	16778520	0.02%
14	268427264	268365880	0.02%

TABLE III
DEVIATION OF TEST LATENCY OVER SEVERAL SIMULATION RUNS

n	Number of Simulation Runs	Latency (L) (Maximum)	Latency (L) (Minimum)	Latency (L) (Average)	Maximum Deviation from Average
10	10	1051053	1046472	1048975	0.24%
12	10	16784258	16773371	16778816	0.03%

most 0.24 percent of the average test latencies for all simulation runs, as shown in Table III. In close agreement with theory, the test latencies arrived at through simulation are very close to the average test latencies given in Table II. If shorter test latencies are required, then techniques for reducing test latency (discussed in [9]) can be employed.

V. HARDWARE FOR CONCURRENT TESTING

In this section we shall discuss the design of a particular type of CBU, which we shall refer to as a *concurrent and off-line observer and tester* (COOT). The basic design philosophy is similar to that used in the design of a BILBO [2]. The COOT has six basic modes of operation. Fig. 4 gives a general design of the COOT. The four building blocks of the COOT are the COOT cell, the feedback function, the scan path multiplexer and the HIT-logic. The COOT is controlled by four independent control inputs NP, TP, SI and FH. The number of control inputs can, however, be reduced but we have chosen four independent control inputs for the clarity of presentation. Each COOT can operate in different modes. The feedback function provides the linear function with a nonlinearity for the exhaustive TG operation. The scan path multiplexer is controlled by SI to open (SI = 1) or close (SI = 0) the feedback path. The HIT-logic is used to generate the signal H_i to indicate the HIT condition for on-line testing. The control input FH is required to force the HIT condition for off-line response compression and scan modes. A design of the COOT cell is given in Fig. 5(a). In this figure, NL represents the normal system latch, TL1 and TL2 are test latches, and ϕ_i and ϕ_j are two nonoverlapping clocks. An implementation of the COOT cell in nMOS is given in Fig. 5(b). This implementation is a modification of the implementation given in [10]. Latches NL and TL2

TABLE IV
OPERATION TABLE FOR THE COOT

NP	TP	SI	FH	Mode of operation
1	1	1	x	Normal Scan Path
0	0	1	1	Test Scan Path
1	1	0	x	Off-line TG
0	0	0	0	On-line TG
0	1	0	1	Off-line RC
0	1	0	0	On-line RC

are realized as dynamic latches while TL1 is realized as a static latch since it is required to hold the data for long duration in on-line test generation and on-line response compression modes. Table IV gives different modes of operations of the COOT for different values of the control inputs. The modes of the COOT are described next in detail.

We must add that not all designs will require existence of all modes mentioned in this paper. Thus it may be possible to simplify the hardware given in this paper. But for the sake of generality we have chosen to treat the most complex case.

1) Normal Scan Path and Off-line Test Generation Modes (NP = 1, TP = 1): In these modes all NL and TL2 latches form a shift register. A complete scan path for off-line scan is formed when $SI = 1$, whereas for $SI = 0$ the feedback path is completed and the COOT acts as an LFSR for off-line test generation. The configuration of the COOT cell for these two modes is shown in Fig. 6(a) by highlighting the configuration path. The normal scan path mode is used to test system latches as well as to read-out and initialize the COOT latches for test generation and response compression modes for off-line testing.

2) Test Scan Path (NP = 0, TP = 0, SI = 1, FH = 1): In this mode all TL1 and TL2 latches form a scan path. The configuration of the COOT cell is shown in Figure 6(b). This mode is used to read-out and initialize the test latches while the system is operating. This mode can also be used to test the test latches of the COOT.

3) On-Line Test Generation Mode (NP = 0, TP = 0, SI = 0, FH = 0): This mode is used for concurrent testing. Configuration of the COOT is shown in Fig. 6(c). The TL1 latch holds the bit to be compared with the content of NL latch. In case of a MISS the signal $H_i = 0$ and the state of the test latches does not change. When a HIT occurs, the signal $H_i = 1$, the test latches of the COOT form an LFSR and a new test vector is generated.

4) Off-Line and On-Line Response Compression Modes (NP = 0, TP = 1, SI = 0): The control input FH is used to distinguish between these two modes. During off-line testing, the tests are applied to the CUT by the TG on every clock cycle, therefore the RC must compress all data. This is achieved by setting $FH = 1$ which in essence sets H_i to 1. In the on-line response compression mode, a response needs to be compressed provided the normal input corresponding to the response was a HIT. Therefore, in this mode the COOT compresses the response only if the COOT acting as an on-line test generator for the CUT

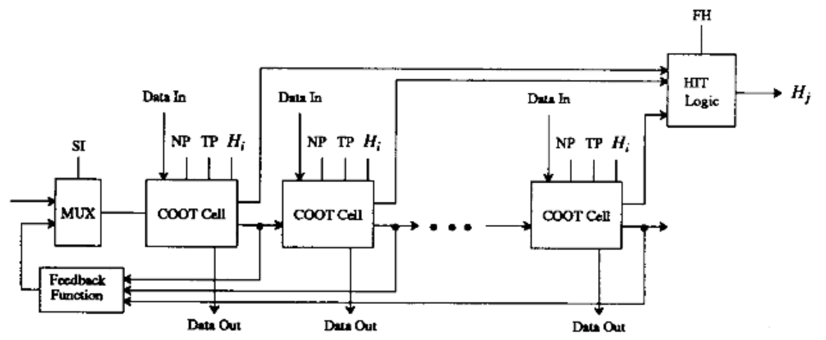
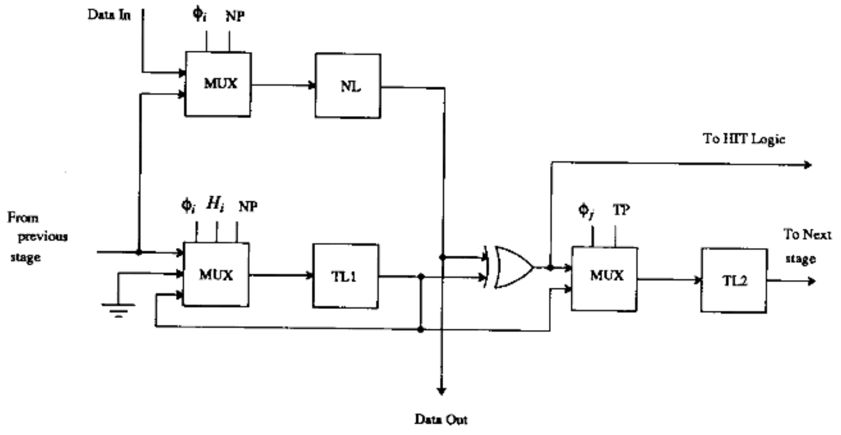
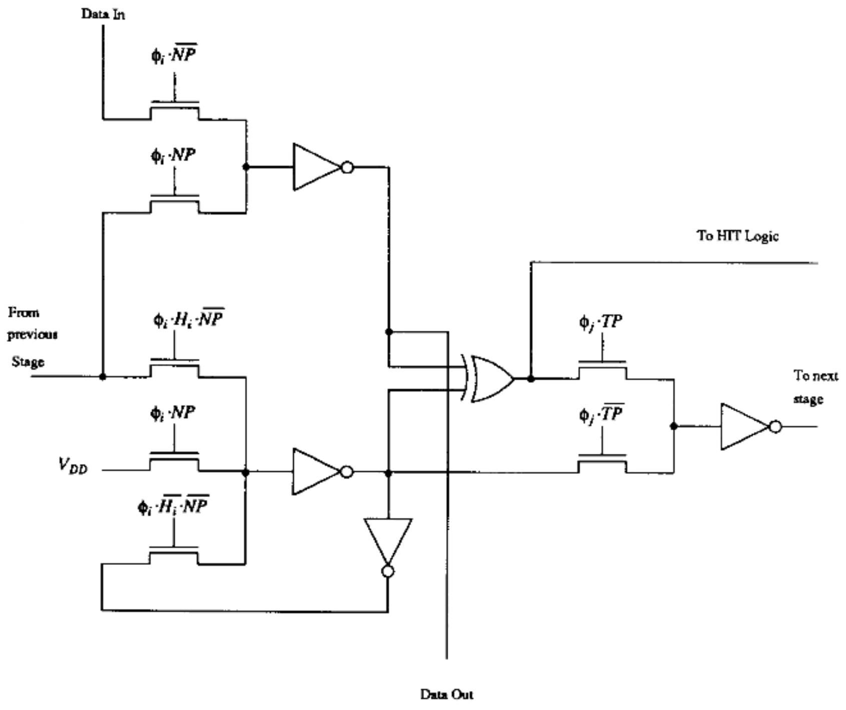


Fig. 4. A general design of the COOT.



(a)


Data In
$\phi_i \cdot \overline{NP}$
$\phi_i \cdot NP$
To HIT Logic
From previous Stage
$\phi_i \cdot H_i \cdot \overline{NP}$
$\phi_j \cdot TP$
To next stage
$\phi_j \cdot \overline{TP}$
V_{DD}
$\phi_i \cdot \overline{H_i} \cdot \overline{NP}$
Data Out

(b)

Fig. 5. A design of the COOT cell. (a) Structure of the COOT cell. (b) An NMOS realization of the COOT cell.

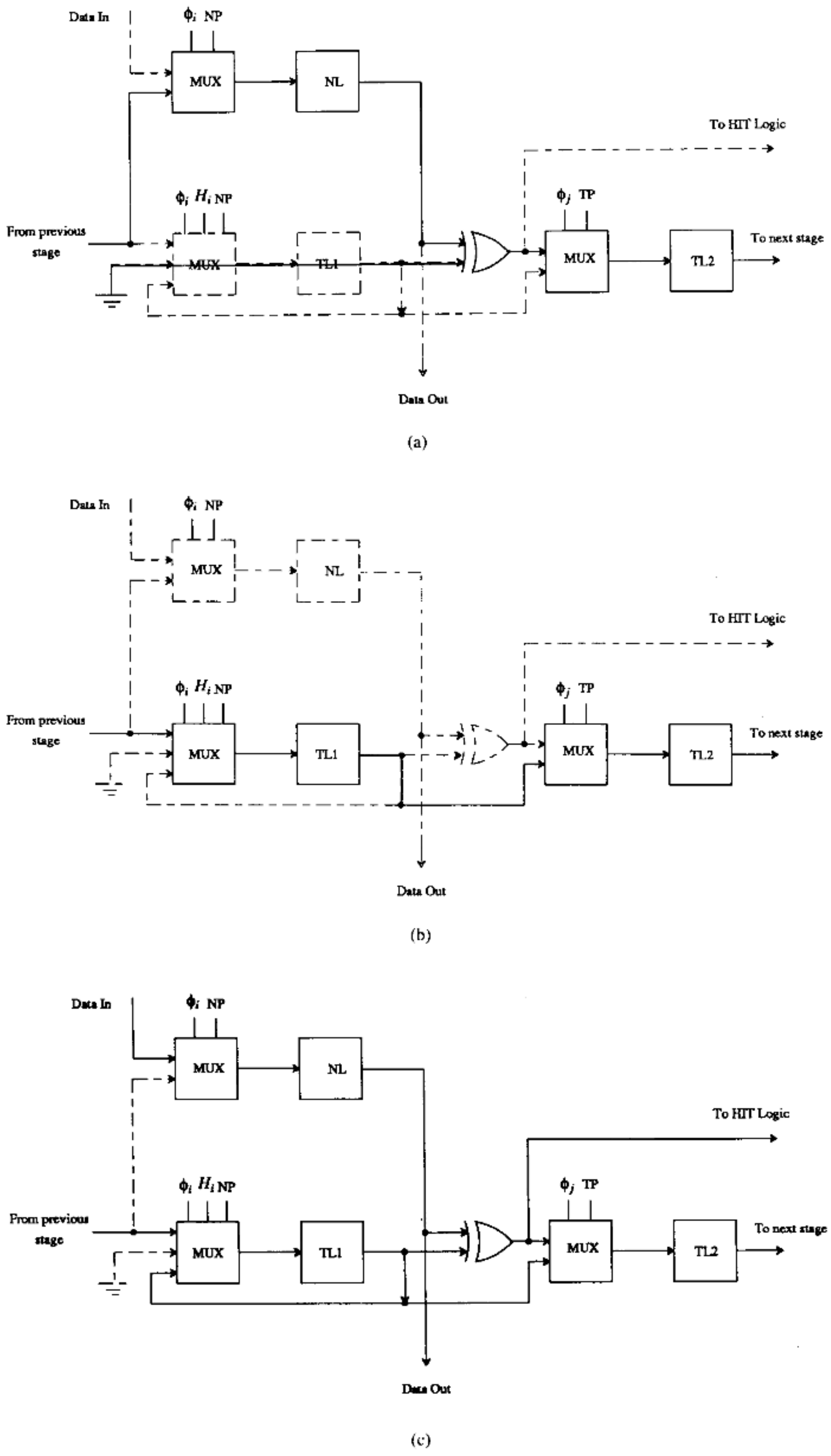
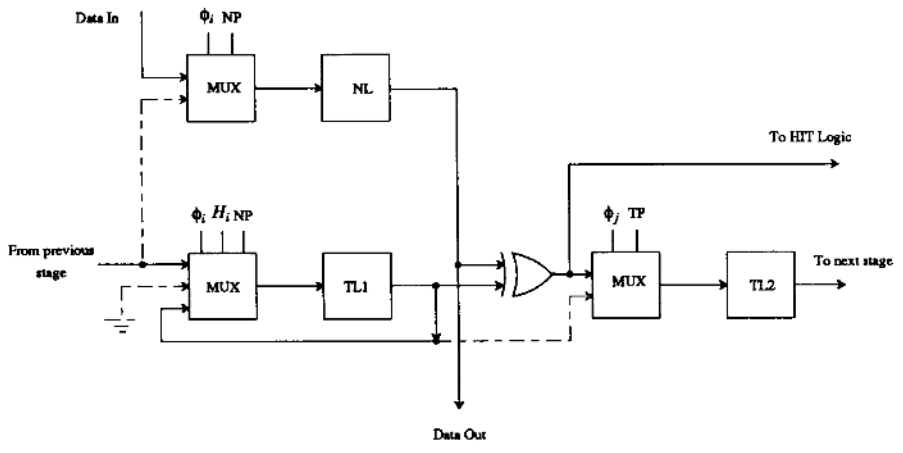


Fig. 6. Different modes of operation of the COOT. (a) Normal scan path and off-line test generation. (b) Test scan path. (c) On-line test generation. (d) On-line and off-line response compression.



(d)

Fig. 6. (*Continued*)

produces a logic 1 on the line H_i. The configuration of the COOT cell for these modes is shown in Fig. 6(d).

A simplified view of the use of the COOT structures in the system operation is shown in Fig. 7. In such an organization off-line testing of all combinational logic structures can be completed in two phases. The method of testing is similar to the testing in a BILBO-oriented test organization [4]. All COOT's are initialized using off-line scan mode before the testing begins. In phase 1 (phase 2) all even (odd) numbered CUT's are tested. Alternative COOTs are configured for off-line test generation and off-line response compression. At the end of the testing all COOT's are read out using off-line scan mode. The compressed responses (signatures) are compared to the expected responses to determine the status of the CUT's. The concurrent testing proceeds in the similar manner but while the system is operating. The COOT's are initialized using the on-line scan mode. Notice that the COOT structure is such that the on-line scan mode has no effect on the normal system latches. To test CUT(i), COOT(i) is used as on-line test generator and COOT($i+1$) is used as on-line response compressor. COOT(i) compares the normal inputs to CUT(i) to its own content. If a normal input to CUT(i) is the same as the content of COOT(i), then it performs two functions: (1) it generates a HIT signal H_i for COOT($i+1$) thereby requiring COOT($i+1$) to record the output of CUT(i) by generating the next signature state; (2) it goes into the LFSR configuration and at the end of the clock cycle its content becomes the next test pattern. Concurrent testing is complete when COOT(i) has gone through all states. A phase of testing is complete when all COOT's in the TG mode have gone through all desired states. At this time all COOT's are read out using test scan path mode. As mentioned earlier the test scan path mode can proceed without interrupting the system operation. The scanned out responses can be compared with the expected responses and the next phase of testing can start. It is also possible to compare the signatures internally without scanning them out in which case only the results of the comparison would need to be read out to determine the status of the circuit.

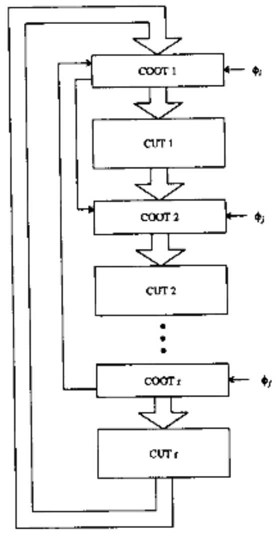


Fig. 7. A simplified test environment for COOT's.

A real system may require more than two phases for completely testing a system as pointed out by Craig *et al.* [11] for off-line testing. The techniques proposed therein for scheduling the tests to reduce total testing time can be used for on-line testing using the COOT structures proposed in this paper. This will also require a design of the controller to completely and correctly carry out the on-line testing function.

VI. Discussion and Conclusions

In this paper we have described a method of testing digital circuits while the system is operating. The resources we use to achieve this are essentially the same as used for off-line testing although depending on the design of off-line BIST, additional logic may be required. As mentioned earlier, in some situations shorter test latencies may be desirable. In such situations the following techniques discussed in [9] can be used.

(1) Circuit partitioning [4].
(2) Verification testing [5].
(3) Roving [12].
(4) Random pattern testing.
(5) Stored tests.
(6) Multiple hardware signature analysis.
(7) Order independent signature analysis [3].

The underlying concept used in these techniques is that the test latency primarily depends upon two factors; (1) the size of the test set T and (2) the probability of a HIT for the inputs applied to the circuit under normal operation. The first five techniques mentioned above reduce the test latency by reducing the number of states of the TG and the last two techniques do so by effectively increasing the probability of a HIT for the normal inputs.

The design of a test resource, proposed in this paper, is such that the resource can be initialized or its status can be read through scan path without interrupting the normal operation of the circuit. Thus the COOT design proposed in this paper can also be used to take a snapshot of the status of the circuit while the system is operating. We should also point out that with the design of the COOT not only the combinational circuit part of the system is tested during on-line testing, but the normal system latches are also tested [10].

This discussion would be incomplete without a comment on the assumptions stated in the Section III. We have assumed in this paper that all inputs to the circuit are equally probable. In a real circuit, not only may this assumption not hold, but some input combinations may never occur during normal operation of the circuit. There are two possible solutions to this problem: (1) we may employ test generators which generate only those test vectors which also occur as normal inputs to the circuit or (2) we may employ time-out indicators, which, if too much time has elapsed for an active test vector, halts the normal system operation and forces a test vector injection. This second method can also be used to reduce the test latency although at the expense of the system performance. Another design of a concurrent tester and its application are studied in [13]. Sharma and Saluja [13] also investigate the use of the concurrent testing technique to: (1) detect and diagnose intermittent faults, (2) detect transient faults, (3) perform system diagnosis, (4) reduce maintenance, and (5) increase system availability.

Finally, the distinction between concurrent testing and self-checking [1] is an important one that has not been emphasized thus far. On-line testing as presented here is targeted at permanent faults whereas self-checking is targeted at both permanent and non-permanent (intermittent and transient) faults. Thus concurrent testing, proposed in this paper, without substantial verification of adequate coverage for non-permanent faults in a particular application is not a total replacement for self-checking. It does guarantee the self-testing property of self-checking, but typically does not provide the fault secure property for nonpermanent faults. Thus, additional hardware or software means are usually needed in combination with concurrent testing to achieve traditional self-checking goals. It should be noted, however, that concurrent testing as proposed here can be used to guarantee that the appropriate input vectors have occurred, an issue which has been a problem with traditional self-testing. Thus the self-testing provided by concurrent testing is stronger than that provided by traditional self-checking designs.

Acknowledgment

The authors are thankful to the referees for providing many helpful and critical comments. The authors especially thank referee 1 whose comments helped improve the presentation substantially.

References

[1] H. Fujiwara, *Logic Testing and Design for Testability*. Cambridge, MA: MIT Press, 1985.
[2] B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-in logic block observation techniques," in *Proc. 1979 Test Conf.*, pp. 37–41, Oct. 1979.
[3] J. Savir, "Syndrome testable design of combinational circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 442–451, June 1980.
[4] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Trans. Comput.*, vol. C-30, pp. 866–875, Nov. 1981.
[5] E. J. McCluskey, "Verification testing—A pseudoexhaustive test technique," *IEEE Trans. Comput.*, vol. C-33, pp. 541–546, June 1984.
[6] J. Losq, "Referenceless random testing," in *Proc. Int. Symp. on Fault-Tolerant Computing*, pp. 108–113, June 1976.
[7] R. A. Frohwerk, "Signature analysis: A new digital field service method," *Hewlett-Packard J.*, pp. 2–8, May 1977.
[8] E. Parzen, *Modern Probability Theory and Its Applications*. New York: Wiley, 1960.
[9] K. K. Saluja, R. Sharma, and C. R. Kime, "Concurrent comparative built-in testing of digital circuits," Tech. Rep. ECE-87-11, Dep. of Elect. Comp. Eng., Univ. of Wisconsin, Aug. 1987.
[10] C. W. Lau and C. R. Kime, "An experimental study of built-in logic block observer (BILBO) designs," Tech. Rep. ECE-86-10, Dep. of Elect. Comp. Eng., Univ. of Wisconsin, 1986.
[11] G. L. Craig, C. R. Kime, and K. K. Saluja, "Test scheduling and control for VLSI built-in self-test," *IEEE Trans. Comput.*, vol. 37, pp. 1099–1109, Sept. 1988.
[12] M. A. Breuer and A. A. Ismaeel, "Roving emulation as a fault detection mechanism," *IEEE Trans. Comput.*, vol. C-35, pp. 933–939, Nov. 1986.
[13] R. Sharma and K. K. Saluja, "An implementation and analysis of a concurrent BIST technique," *Int. Symp. on Fault-Tolerant Computing*, pp. 164–169 June 1988.

*

Kewal K. Saluja (S'70–M'73) received the B.E. degree in electrical engineering from the University of Roorkee, Roorkee, India, in 1967, and the M.S. and Ph.D. degrees from the University of Iowa, Iowa City, in 1972 and 1973, respectively.

He is currently an Associate 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, VLSI design, and testing. Previously he was at the University of Newcastle, NSW, Australia. He has also held visiting and consulting positions at number of institutions such as the University of Southern California, University of Iowa, and Hiroshima University. His research interests include design for testability, fault-tolerant computing, VLSI design and computer architecture.

Rajiv Sharma received the B.E. degree in electronics and electrical communications from the Punjab Engineering College, Chandigarh, India, in 1985, and the M.S. degree in electrical engineering from the Michigan Technological University, Houghton, in 1986.

He was with Michigan Technological University, as a Teaching Assistant from 1985 to 1986, and as a Research Assistant at the University of Wisconsin-Madison from 1986 to 1988. He is currently working in the area of computer-aided test products development with Gateway Design Automation Corporation, Lowell, MA. His research interests include design for testability, VLSI design, fault tolerant computing, computer architecture, and design automation.

Charles R. Kime (S'60–M'66–SM'79) received the B.S. degree from the University of Iowa, Iowa City, in 1962, the M.S. degree from the University of Illinois, Urbana, in 1963, and the Ph.D. degree from the University of Iowa in 1966, all in electrical engineering.

He joined the faculty of the University of Wisconsin-Madison in 1966 where he is currently a Professor in the Department of Electrical and Computer Engineering. He spent the 1973–1974 academic year as a Visiting Associate Professor in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. His current research and teaching interests are in testing and built-in test, VLSI systems design, fault-tolerant computing, computer architecture and logic design.

Dr. Kime is a member of the Association for Computing Machinery, the American Society of Engineering Education, and Sigma Xi. He has served as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS and as the General Chairman of the 1979 International Symposium on Fault-Tolerant Computing.