Fig. 7. Normalized CPU time as a function of the number of gates. The data have been normalized with respect to the case of the largest circuit (C7552). Symbols $\circ$ and ∇ denote the fault free and heuristic simulation time, respectively.

TABLE II
COVERAGE OBTAINED FOR DETERMINISTIC TEST SEQUENCES

Circuit	Test Patterns	C_{exact}	C_{crit}	$ C_{\text{exact}} - C_{\text{crit}} $
C432	93	98.80	98.68	0.12
C499	70	99.13	99.02	0.11
C880	109	100.00	99.81	0.19
C1355	112	99.70	98.78	0.92
C1908	185	99.71	99.58	0.13
C2670	190	96.35	96.33	0.02
C3540	233	96.33	95.28	1.05
C5315	208	99.34	99.34	0.00
C6288	51	99.46	99.46	0.00
C7552	247	96.95	97.14	0.19

ACKNOWLEDGMENT

The author would like to thank one of the reviewers of his significant contribution in improving the early version of this paper and, in general, all the reviewers for their helpful comments and suggestions. Furthermore, the authors are grateful to Dr. A. Liroy of Polytechnic of Turin, Italy, for providing the deterministic test patterns used in this work.

REFERENCES

- [1] M. Abramovici, P. R. Menon, and D. Miller, "Critical path tracing: An alternative to fault simulation," *IEEE Design Test.*, vol. 1, pp. 83-92, Feb. 1984.
- [2] K. Antreich and M. Schulz, "Accelerated fault simulation and fault grading in combinational logic circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 704-712, Sept. 1987.
- [3] M. Favalli, P. Olivo, M. Damiani, and B. Riccò, "Fault simulation of unconventional faults in CMOS circuits," to be published.
- [4] P. Menon, Y. Levendel, and M. Abramovici, "Critical path tracing in sequential circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1988, pp. 162-165.
- [5] F. Maamari and J. Rajski, "A fault simulation method based on stem regions," in *Proc. IEEE Int. Conf. Computer Aided Design*, 1988, pp. 170-173.
- [6] W. Ke, S. Seth, and B. Bhattacharya, "A fast fault simulation algorithm for combinational circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1988, pp. 166-169.
- [7] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," in *Proc. IEEE Int. Symp. on Circuit and Systems*, 1985, pp. 663-698.

- [8] A. Liroy, "Adaptive backtrace and dynamic partitioning enhance ATPG," in *Proc. IEEE Int. Conf. on Computer Design*, 1988, pp. 62-65.

A Method of Reducing Aliasing in a Built-In Self-Test Environment

Keiho Akiyama and Kewal K. Saluja

Abstract—In this paper we propose a method of reducing aliasing in built-in self-test of VLSI circuits. The method is based on the use of transition count testing. We first give a new formulation of the problem in terms of finding a test generator as opposed to solving the problem at the data compaction end. An algorithm is proposed which can be used to find a counter-based test pattern generator. This test generator tests a circuit exhaustively or pseudo-exhaustively such that the aliasing is reduced substantially provided the data compactor used is a transition counter. Experimental results are presented to substantiate these claims.

Index Terms—Built-in self-test (BIST) transition count testing, VLSI testing, compact testing signature analysis.

I. INTRODUCTION

Some of the problems that confront the conventional testing methods are: 1) excessive time required for test pattern generation, 2) the external testers are complicated and expensive, 3) test application time can be very high, and 4) limited number of inputs, outputs and/or test pins make it difficult to test and observe the entire circuits from outside.

A built-in self test (BIST) design offers a practical solution to these problems in some environments. BIST design employs an on-chip test circuit. Test patterns are generated by an on-chip test pattern generator (TPG) and the results are evaluated by an on-chip response evaluator (RE). BIST has its own advantages and disadvantages. It does not need the time consuming test pattern generation process, nor does it require the storage of all test vectors. On the other hand, it may not be able to detect all faults since not all the test results are evaluated. Instead of evaluating test results one by one, BIST evaluates the compacted data which is obtained by some compaction technique after the entire sequence of test vectors is applied. This may result into the masking of errors in the output sequence during the compaction process. This is often termed as aliasing. Consequently, due to aliasing, some faults may remain undetected.

Several data compaction methods (see [1] for references) such as transition count testing [2], signature analysis [3], and syndrome testing [4], have been proposed for test results compaction. The

Manuscript received March 30, 1989; revised February 16, 1990. This work was supported in part by the University of Wisconsin Graduate Research Committee and by the National Science Foundation under Grant MIP 8509194. This paper was recommended by Associate Editor S. C. Seth.

K. Akiyama was with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI. He is now with the Language Application Systems Program Department, Hitachi Software Engineering, Co., Yokohama, Japan.

K. K. Saluja is with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706.

IEEE Log Number 9042074.

¹In this paper we will use the word *compaction* as opposed to *compression* in conformity with the definition of this term presented in [18] by McCluskey.

aliasing probability for methods employing signature analysis can be evaluated by using the techniques proposed in [5], [6]. Robinson and Saxena [7] have presented a unified view for analyzing different methods of compaction. Techniques have also been proposed to reduce aliasing probability by using large data compactors [8], [9], multiple signatures [1], [10], [11], and output data modification [12]. However, none of these methods can completely avoid aliasing² and all of these methods have significant space and/or time overhead associated with them. In this paper we propose a method based on transition count testing which is suitable for the BIST environment, and also reduces the aliasing substantially.

The transition count testing method counts the occurrences of transitions, termed as *transition count* (TC), in the output sequence during testing. The probability of aliasing depends on the TC value [13], [14] implying that for certain TC values the probability of aliasing can be lower than for other TC values for a given circuit. Further, the TC value for a circuit can be controlled by the order of application of the test vectors. Thus our objective is to find a TPG for a given circuit, which will generate the test vectors in an order such that it renders a TC value causing low aliasing. Further, we would want that the aliasing due to compaction by TC should be lower than the compaction by other methods.

In this paper, after over-viewing the basic features of BIST, we formulate the TC testing problem in Section II. The TC testing in a BIST environment is discussed in Section III. An algorithm to generate a configuration of the TPG, which orders test vectors for TC testing such that aliasing occurs with a smaller probability compared to signature analysis methods, is presented in Section IV. In Section V, experimental results are given to establish the effectiveness of the algorithm.

II. BIST OVERVIEW AND PROBLEM FORMULATION

BIST consists of three components: a TPG, the *circuit under test* (CUT) which is usually combinational, and an RE. The basic BIST structure is shown in Fig. 1. A control signal enables and disables the TPG and the RE. In the normal mode of operation, the input signals from the other regions of the chip go through the TPG and reach the CUT, while the output signals go through the RE and reach their destinations. In the test mode, the CUT is separated from the other parts of the chip, and is tested independently by using the TPG and the RE. The TPG feeds a sequence of test vectors to the CUT. The RE evaluates the CUT's responses to these test vectors, and reports the result(s) to the outside world.

A simple way to realize the TPG and the RE is to use a ROM. However, this can have only a limited practical application because of its high overhead. We shall restrict our attention to a combinational circuit using a TPG which applies exhaustive or pseudo-exhaustive test patterns to the CUT. Although, exhaustive test patterns can be generated by counters and/or by *linear feedback shift registers* (LFSR's), we shall consider only a counter-based TPG in this paper. The RE will be assumed to be a transition counter which can be realized by a filter followed by a counter or a LFSR [7].

The TC value is the number of times the output value of the CUT changes during the course of the test. The TC testing method was not originally devised for BIST, but as a means to make conventional testing efficient. The following arguments were used by Hayes [2] to demonstrate the effectiveness of TC testing for single output circuits in conventional testing environment. Let t_i be a test vector whose fault-free response is r_i (r_i is either 0 or 1), and $T = t_1, t_2, \dots, t_m$ be a set of test vectors. In conventional testing, all the fault-free responses, namely $R = r_1, r_2, \dots, r_m$ are stored in

²Complete avoidance of aliasing is possible by the methods listed earlier only in the extreme/trivial circumstances. For example, if the data compactor is large enough to store the complete output sequence, or if the output data modifier can cancel every "one" in the output sequence, than no aliasing will take place. However, such conditions are often too costly to be used in practice, especially in a BIST environment.

Fig. 1. Basic BIST configuration.

the external memory. Faults are detected by comparing the responses of the CUT to the expected responses. For TC testing purposes, the test set T is applied in a sequence $S^* = t_p^0, t_1^0, t_2^0, \dots, t_p^1, t_q^1, t_1^1, t_2^1, \dots, t_q^1$, where $p + q = m$ and t_i^0 is a test vector from the test set T whose fault-free response is 0. During testing, the sequence S^* is used to apply test vectors and instead of observing the complete output response sequence a transition counter is used as an RE. Note that the RE based on TC consists of a "1-b" transition counter (with overflow detection logic). Hayes [2] proved that all faults detected by a conventional testing method will also be detected by the above TC test if the circuit is realized as a two level circuit.

It is to be noted that in terms of aliasing, exactly one error pattern, in which every bit of the response sequence is complemented due to a fault, is aliased and all other patterns are detected by the TC test. This argument suggests that the TC testing is a useful RE provided the test vectors can be ordered in such a way that only a small number of error sequences are masked. This will also imply a low aliasing probability in TC testing environment. In the BIST environment, with a counter or an LFSR as a TPG, it may be impractical to arrange the test vectors as they appear in the sequence S^* described earlier. However, by arranging the connections between the TPG and the CUT properly, the application of the order of test vectors can be changed. This can potentially reduce the probability of aliasing, thus making TC testing possibly a very effective RE.

Further, if the error-free TC value is very small, then only a few counter bits are needed for the TC-based RE. For example, for a 10 input circuit, the maximum possible TC value is 1023, requiring a 10-b counter in the RE. However, if the error-free TC value is small, say 25, then only a 5-b counter (with overflow detector) is required in the RE, while the effectiveness in terms of the probability of aliasing is still as good as for a 10 b or a larger counter in the RE. Thus we need to investigate ways to implement a TPG such that TC testing can be used as a more effective RE than other compactors.

III. TC TESTING IN BIST ENVIRONMENT

As mentioned in the previous section, for effective use of the TC testing, test vectors must be generated in a certain order. Below, we briefly describe the methods used to compare the effectiveness of the TC testing with the syndrome testing and the signature analysis. Further, we shall initially restrict our attention to single output circuits.

3.1. Aliasing Problem in TC Testing

We use three different approaches to show that appropriate choices of TC values result into fewer error maskings relative to

signature analysis. In the discussion below, we shall assume that an n -input function f has a TC value t when tested by an exhaustive test sequence (of length 2^n) in a given order. The total number of sequences with TC value equal to t is $N(t) = \binom{2^n-1}{t} \times 2$.

First, let us consider the simple case and assume all errors are equally likely. In this case the probability of aliasing for the function f is $P(t) = (\binom{2^n-1}{t} \times 2 - 1) / (2^{2^n} - 1)$. This implies that for very large values of t , close to $(2^n - 1)$, or for very small values of t , close to 1, the TC testing is more effective than signature analysis. For example, for $n = 7$, the TC test is considered more effective than a 7-b LFSR provided $t \leq 51$ or $t \geq 76$.

Second, let us consider the argument advanced by Zorian and Agarwal to determine the deception volume [12]. For a given value of t , the deception volume is $D(t) = (\binom{2^n-1}{t} \times 2 - 1)$. As t becomes small, $D(t)$ becomes small, causing fewer errors, and hopefully fewer faults, to be masked, irrespective of error probabilities. Arguments analogous to those in [12], can be used to show that our algorithm (to be presented in Section IV) reduced the deception volume substantially by reducing the value of t .

Third, using the arguments similar to those used by Robinson and Saxena [7], we can determine the equivalent length Le of an LFSR that provides the same aliasing as TC testing. For a test sequence of length 2^n and TC value t , $Le = 2^n - \log_2 \binom{2^n-1}{t}$. Once again, for a small value of t the equivalent length Le is much larger than $\log_2 t$.

All the above arguments support the fact that a very small value (or a very large value) of t will result into reduced error masking irrespective of error probabilities. We must comment that a similar conclusion is drawn by Zorian and Agarwal [12] in respect of syndrome of the output data stream. Their approach was then to reduce the number of 1's in the output data stream by adding extra logic to modify the output data stream. Our approach is to re-organize the output data stream by changing the order of test vectors so that the resulting TC value is very small or very large. A reason for attempting to make the TC value small, as opposed to large, is given in the following subsection.

3.2. Arrangement of the Test Order in BIST Environment

The TC value, t , for an n -variable nontrivial function with k minterms is bounded as $1 \leq t \leq 2 \times \min(k, 2^n - k)$. The minimum TC value, $t = 1$, occurs when test vectors are ordered such that all tests whose expected responses are ones are applied prior to tests whose expected responses are zeros, or vice versa. We have also discussed that the aliasing probability in TC testing is reduced, provided the TC value is very small or very large. Thus TC testing can be made more effective provided the TC value by the chosen TPG is very small or very large. Furthermore, it is noted in [7] that for most actual functions the number of minterms k is small. Hence, the TC value is expected to be relatively small for most practical functions. Therefore, it is reasonable to try to make the TC value "small" (as opposed to "large") by choosing a TPG appropriately.

For the exhaustive testing of an n -input function, there are $2^{n!}$ ways of ordering the test vectors. Further, the output sequence consists of k ones and $(2^n - k)$ zeros. The total number of distinct sequences containing k ones and $(2^n - k)$ zeros is $\binom{2^n}{k}$. It can be shown that each of the distinct sequences, which contains exactly k ones and $(2^n - k)$ zeros, occurs $k!(2^n - k)!$ times in the $2^{n!}$ possible sequences. If all k ones are either at the head or the tail of the sequence, TC value is one. This occurs in $2 \times (k!(2^n - k)!)$ ways.

In a BIST environment, if the TPG is implemented by an LFSR or a counter, such an ordering of test vectors may not be possible to attain. Below we describe three methods which are suitable for use in BIST environment and will be considered in this paper to rearrange the test vectors. The example of Fig. 2 will be used to explain the methods.

³We denote factorial m by $m!$. Also, $m! / k!(m - k)!$ will be denoted by $\binom{m}{k}$.

Fig. 2. Arrangements to reorder test vectors. (a) No arrangement. (b) Permutation. (c) Complementation. (d) Initialization.

- 1) **Permutation:** The order of test vectors is changed by rearranging the connections between the outputs of the TPG and the inputs of the CUT. The effect of this method for a 3-b TPG is shown in Fig. 2(b).
- 2) **Complementation:** Either the complemented or the uncomplemented outputs of the TPG can be connected to the CUT. Clearly, if some of the complemented outputs of the TPG are connected to the CUT, while leaving the remaining connections to the uncomplemented outputs of the TPG, the order of test vectors will be altered. This situation is depicted in Fig. 2(c) for a 3-input CUT. In this figure, the effect on the order of test sequence by connecting the complements of bit b_0 and b_2 of the 3-b TPG to the CUT, while using the bit b_1 as uncomplemented, is shown.
- 3) **Initialization:** Finally, a small change in the order of the tests can be achieved by starting the counter (TPG) in a state other than the all-zero state. This situation is shown in Fig. 2(d).

Combinations of these three methods, termed as *arrangements* in sequel, generate a variety of test orderings. Although only a small fraction of all possible test orders are realizable using arrangements [14], experimental results show that in most cases, near optimum solutions (very small TC value) can be obtained from these limited choices [14].

It is expected that the implementation of arrangements will require only small overhead. For example, for a permutation, no extra logic is needed since it merely changes the routing. Similarly, complementary outputs can be extracted without extra inverters if each memory element of the TPG is realized as a flip-flop or as a latch which provides both Q and $\bar{Q}$. Initialization can be changed

without extra cost. All the above comments hold for a TPG implemented by a LFSR or a counter. If a counter is chosen as a TPG, the upper bounds on the value of TC by using the permutation and complementation methods are given in [14].

3.3. TC Testing for Multiple Output Circuits

A CUT with m ($m > 1$) outputs can be considered either as a single output circuit with all the outputs of the CUT processed through a space compressor [16], or as m single output circuits.

In the first case, all m outputs are exclusive-ored (a linear space compressor [16]), and the output of the space compressor is fed to the RE (transition counter). The TPG is arranged such that the TC value of the exclusive-ored output is small. While this idea is simple, the cost of the exclusive-or tree can be prohibitive.

In the second case, there are several ways to handle the problem. Three possible ways to treat an m output circuit are as follows.

- i) For each output of the circuit provide a separate TPG. Test one output of the CUT at a time and multiplex the corresponding TPG.
- ii) Use a single reconfigurable TPG to generate test vectors in a desired order for each output.
- iii) Use only one TPG. The TPG should provide the test vectors in a fixed order such that the TC value at each of the m outputs of the CUT is "small."

Among these, the first method is complete but costs too much. The second method is likely to have smaller overhead than the first but the control logic can be very complicated to change the TPG to desired configurations. The third method is likely to have the least area overhead.

For an RE of a multiple output CUT, two possible implementations are: i) m separate counters and ii) one counter and a multiplexer. All outputs are evaluated concurrently in i), while outputs are evaluated sequentially in ii). In i), the area penalty is high but the testing time is low, whereas in ii), the area penalty is low but the test completion time is large.

IV. AN ALGORITHM TO GENERATE TC TPG

In this section, we present an algorithm which makes use of heuristics to make the TC testing more effective in a BIST environment. To arrive at an appropriate arrangement, the algorithm makes use of the sum-of-products expressions for the CUT. In the algorithm, it is assumed that a counter is used as an exhaustive TPG. Thus the goal is to find an arrangement for ordering the test vectors such that TC value for the CUT is as small as possible. The algorithm, in its present form, is well suited for those circuits which have been partitioned for pseudo-exhaustive and exhaustive testing. The algorithm can be extended to include a random testing BIST environment although the results may not be as strong as presented in this paper. We conjecture that even in random testing BIST environment TC testing is likely to outperform other compaction methods.

The basic strategy for arriving at a good arrangement is as follows. In order to make the TC value small, the rows of a truth table are to be ordered such that ones in the output column appear in consecutive rows. Suppose an n -variable function f contains a product term with i variables. This product term covers 2^{n-i} minterms. By connecting all i variables in the product term to the i most significant bits of the counter, all 2^{n-i} minterms corresponding to the product term can be made to appear in the consecutive rows in the truth table. For instance, for a four input function $f = \bar{c}d + ab\bar{c} + \bar{a}bcd$, the product term $\bar{c}d$ covers four minterms. If c and d are connected to b_3 and b_2 (the two most significant bits of a 4-b counter), or vice versa, then all four minterms covered by $\bar{c}d$ will appear in the four consecutive rows in the truth table. Furthermore, if c is connected to b_3 (b_2), and d is connected to b_2 (b_3), then the four minterms appear in the last four rows in the truth table. Thus the transition caused by this product term occurs

only once (from 0 to 1). This suggests that by assigning more significant bits of the TPG to the variables which appear in the product term with fewer variables, and by connecting complemented output of the counter bit to the input whose complement appears in the product term, the value of the transition count for the function under consideration can be reduced. The algorithm achieves this by constructing a *variable weight table*. The algorithm, in Pascal-like notation, is given below. Implementation details of the algorithm can be found in [14].

Algorithm TC_Gen;

```

{
  input:   Boolean expression of the function
  output:  counter bit assignment for the function
  notation: function is expressed as minimum
           sum-of-products consisting of
           (i)  $n$  distinct variables,  $v_1, v_2, \dots, v_n$ ,
           (ii)  $k$  product terms,  $p_1, p_2, \dots, p_k$ ,
           (iii)  $l_i$  variables in a product term  $p_i$ .
}
var   W: array [1 .. 2, 1 .. n] of integer;
      Order: array [1 .. n] of integer;

Procedure Initialize_W;
begin;
  assign 0 to all elements of  $W[i, j]$ ;
end;

Function TermWeight(i); { calculates the weight value of  $p_i$  }
begin;
  TermWeight(i) =  $2^{(n-l_i)}$ 
end;

Procedure Fill_W; { prepares the weight table  $W$  }
begin;
  for  $i := 1$  to  $k$  do
    for  $j := 1$  to  $n$  do
      begin
        if product term  $p_i$  contains the
          variable  $v_j$  then
          add TermWeight(i) to  $W[1, j]$ ;
        if product term  $p_i$  contains
          complement of variable  $v_j$  then
          add TermWeight(i) to  $W[2, j]$ ;
      end
    end;
end;

Procedure Ordering; { assign the order value to
                    Order[i] }
begin
  for  $i := 1$  to  $n$  do
    begin
      set Order[i] := order-value provided
      max ( $W[1, j]$ ,  $W[2, j]$ ) is order_valueth
      largest among the max ( $W[1, j]$ ,
       $W[2, j]$ )
      where  $1 \leq j \leq n$ .
      { note that each Order[i] is assigned
      a different value }
    end
  end;

end;

Procedure Assign_Counter_Bits;
begin
  Ordering;
  for  $i := 1$  to  $n$  do
    if  $W[1, i] \geq W[2, i]$  then
      connect uncomplemented Order[i]th
      output bit of the counter (the TPG) to
      input  $v_i$  of the circuit else
      connect complemented Order[i]th
      output bit of the counter to the input
       $v_i$  of the circuit
  end;
end;
```

Fig. 3. CAD system for BIST.

TABLE I
EXPERIMENTAL RESULTS

PLA	Specifications			Results		
	Inputs	Outputs	Product Terms	Win/Lose	Number of Bits for RE	<i>Le</i>
z5xp1	7	10	65	9/1	7	65.9
z9sym	9	1	84	1/0	7	151.9
add6	12	7	355	6/1	11	1834.9
adr4	8	5	75	5/0	8	104.5
alu1	12	8	19	8/0	11	1767.3
alu2	10	8	68	8/0	10	408.6
alu3	10	8	66	7/1	9	611.1
apla	10	12	25	12/0	7	778.4
col4	14	1	14	1/0	5	16 104.4
dc1	4	7	9	3/4	3	3.8
dc2	8	7	39	7/0	7	55.2
dist	8	5	120	5/0	8	103.8
dk17	10	11	18	11/0	7	806.5
dk27	9	9	10	9/0	5	419.5
dk48	15	17	21	17/0	11	28 539.7
f51m	8	8	76	7/1	8	143.1
gary	15	11	107	11/0	9	31 229.1
in0	15	11	107	11/0	9	31 229.1
in1	16	17	104	17/0	10	62 157.1
mlp4	8	8	127	5/3	7	72.4
pla	8	8	127	5/3	7	72.4
radd	8	5	75	4/1	7	78.4
rd53	5	3	31	3/0	5	6.4
rd73	7	3	127	3/0	7	13.2
risc	8	31	28	31/0	4	233.6
root	8	5	57	5/0	4	218.6
sqn	7	3	38	3/0	6	28.7
sqr6	6	12	50	8/4	6	19.1
temp	5	3	31	3/0	5	6.4
tial	14	8	579	8/0	13	5394.6
wim	4	7	9	4/3	3	5.0
z4	7	4	59	3/1	7	35.5
average	9.1	8.2	85.0	7.5/0.7	7.2	5523.7

```

begin                               { main procedure }
    Initialize_W;
    Fill_W;
    Assign_Counter_Bits;
end.

```

The algorithm stated above generates an arrangement for a single output function. To find a good arrangement for a multiple output function, the algorithm is used a number of times to find an arrangement for each of the outputs, independently. Then, an arrangement which is effective for most of the outputs is chosen. The algorithm was implemented as a program and integrated with Berkeley CAD tools as shown in Fig. 3. The CAD system can be extended to accept circuits described by netlists interconnecting different gate types, by incorporating a simulator in the tool set. The runtime of the TC_GEN compared to other CAD tools, such as Espresso [17] and Eqntott, for the circuits considered in our experiments (see Section V), was negligibly small.

V. EXPERIMENTAL RESULTS

As an experiment, the algorithm was tried on 32 PLA's. The algorithm TC_GEN was used to find a TPG for each of these PLA's. For every PLA and the corresponding TPG provided by the algorithm, the value of the TC was determined at each output of the PLA. For each of the TC values, the aliasing probability $P(t)$, defined in Section III, was computed. The value of $P(t)$ was compared with the probability of aliasing due to the use of signature analyzer (LFSR) for every output of the PLA. We also computed the equivalent length, Le , as defined in Section III for each output of the PLA's.

The results are summarized in Table I. The results are compared by assuming that the signature analysis method uses an r -bit signature analyzer, where r is the number of inputs to the CUT. The reason for choosing r as the length of the LFSR was that the TC testing technique will never need the length of the compactor to be larger than r . In Table I, the entries in the column "number of bits of RE" are determined from the maximum TC value for any of the outputs of the PLA. These values also include 1 b required to implement the filter and 1 b to store the overflow in RE. Thus for the circuit **dk48** with 15 inputs and 17 outputs, the RE will have no more than 11 b in it to meet the worst-case TC condition. Note that it means the maximum value of TC was less than 2^{11-2} for each of the 17 outputs of this circuit.

In the "win/lose" column, the "wins" indicate the number of outputs for which the TC value was low enough for the TC test to outperform, in terms of aliasing probability, the signature analysis method. For **dk48**, the TC value for all outputs was such that it outperformed signature methods. Whereas, for the circuit **alu3** with 10 inputs and 8 outputs, the transition count value at 7 out of 8 outputs for the TPG provided by TC_GEN was substantially low, and for only one of the outputs the value of transition count was high enough where LFSR may have performed better than TC testing.

On the average, for 91.3% of the 240 outputs, the proposed method gives better results than the signature analysis. Also, TC-based RE required 20% fewer number of bits in the output compactors compared to use of the LFSR's. The comparison of the effectiveness of the method in terms of equivalent length Le is even more impressive. In general, it is evident from this table that in

almost all cases TC testing, as proposed in this paper, is likely to cause substantially lower aliasing than signature analysis employing an LFSR.

VI. CONCLUSIONS

In this paper we have shown that TC testing, which has not been considered as a compaction technique for BIST, can be made effective by arranging the order of test vectors. Near optimum ordering of the test vectors can be obtained by using a counter as a TPG. An algorithm to arrive at a good design of a counter-based TPG, using sum-of-products expressions for a CUT, is proposed. Experimental results show that for more than 90% of the cases considered in the paper, the algorithm finds arrangements that make TC testing more effective than the signature analysis.

REFERENCES

- [1] J. S. Upadhyaya and K. K. Saluja, "Signature techniques in fault detection and location," in *Spectral Techniques in Fault Detection*, M. Karpovsky, ed. Orlando, FL: Academic, 1986.
- [2] J. P. Hayes, "Transition count testing of combinational logic circuits," *IEEE Trans. Comput.*, vol. C-25, pp. 613-620, June 1976.
- [3] R. A. Frohwerk, "Signature Analysis: A new digital field service method," *Hewlett-Packard J.*, pp. 2-8, May 1977.
- [4] J. Savir, "Syndrome-testable design of combinational circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 442-451, June 1980.
- [5] T. W. Williams, W. Daehn, M. Gruetzner, and C. W. Starke, "Aliasing errors in signature analysis registers," in *Proc. Int. Test. Conf.*, Sept. 1987, pp. 637-644.
- [6] A. Ivanov and V. K. Agarwal, "An iterative technique for calculating aliasing probability of linear feedback shift registers," in *Proc. Int. Symp. on Fault Tolerant Computing*, June 1988, pp. 70-75.
- [7] J. P. Robinson and N. R. Saxena, "A unified view of test compression methods," *IEEE Trans. Comput.*, vol. C-36, pp. 94-99, Jan. 1987.
- [8] J. E. Smith, "Measure of effectiveness of fault signature analysis," *IEEE Trans. Comput.*, vol. C-29, pp. 510-514, June 1980.
- [9] N. R. Saxena and J. P. Robinson, "Accumulator compression testing," in *Proc. Int. Symp. on Fault Tolerant Computing*, June 1985, pp. 300-305.
- [10] S. Z. Hassan and E. J. McCluskey, "Increased fault coverage through multiple signatures," in *Proc. Int. Symp. on Fault Tolerant Computing*, June 1984, pp. 354-359.
- [11] D. K. Bhavsar and B. Krishnamurthy, "Can we eliminate fault escape in self testing by polynomial division (signature analysis)," in *Proc. Int. Test Conf.*, Oct. 1984, pp. 134-139.
- [12] Y. Zorian and V. K. Agarwal, "A general scheme to optimize error masking in built-in self-test," in *Proc. Int. Symp. on Fault Tolerant Computing*, June 1986, pp. 410-415.
- [13] P. H. Bardell, W. H. McAnny, and J. Savir, *Built-In Pseudorandom Testing of Digital Circuits*. New York: Wiley, 1988.
- [14] K. Akiyama, "Transition count testing in BIST environment," M.S. Res. Report, Dep. Comput. Sci., Univ. of Wisconsin-Madison, May 1987.
- [15] S. K. Gupta and D. K. Pradhan, "Combining data compression techniques," presented at BIST Workshop, Charleston, SC, Apr. 1987.
- [16] S. M. Reddy, K. K. Saluja, and M. G. Karpovsky, "A data compression technique for built-in self-test," *IEEE Trans. Comput.*, vol. 37, pp. 1151-1156, Sept. 1988 and *IEEE Trans. Comput.*, vol. 38, p. 320, Feb. 1989.
- [17] R. K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic, 1984.
- [18] A D&T Roundtable, "Built-in self-test: Are expectations too high?," *IEEE Design Test Comput.*, pp. 66-74, June 1989.