On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits

Seiji Kajihara
Dept. of Computer Sciences and Electronics, and
Center for Microelectronics Systems
Kyushu Institute of Technology
Iizuka 820-8502 Japan

Kohei Miyase
Dept. of Computer Sciences and Electronics
Kyushu Institute of Technology
Iizuka 820-8502 Japan

Abstract

Given a test set for stuck-at faults, some of primary input values may be changed to opposite logic values without losing fault coverage. We can regard such input values as don't care *(X). In this paper, we propose a method for identifying X inputs of test vectors in a given test set. While there are many combinations of X inputs in the test set generally, the proposed method finds one including X inputs as many as possible, by using fault simulation and procedures similar to implication and justification of ATPG algorithms. Experimental results for ISCAS benchmark circuits show that approximately 66% of inputs of un-compacted test sets could be X in average. Even for compacted test sets, the method found that approximately 47% of inputs are X. Finally, we discuss how logic values are reassigned to the identified X inputs where several applications exist to make test vectors more desirable.*

1. Introduction

Research on test generation for logic circuits has been done for a long time and many significant results have been obtained [1]. The traditional tasks of test generation were to achieve high fault coverage (sometimes it means complete fault efficiency) for single stuck-at faults with short computing time. Then, with the progress of VLSI technology, some additional features have been required to generated test sets as follows:

(1) Not only 100% fault efficiency for single stuck-at faults, but also defects modeled by other faults such as bridging faults, delay faults should be detected [2].
(2) The number of test vectors should be small due to reduction of test application time and memory limits of LSI testers [3].
(3) Power dissipation during testing should be reduced [4].

Once a test set was generated, it may be difficult to modify them so as to satisfy the above requirements because every primary input value of test vectors in the test set has been specified to either 0 or 1. It is obvious that test vectors generated for random pattern fault simulation don't contain Xs. Even for test vectors generated by a deterministic test generator such as SOCRATES [5], the final test vectors don't contain Xs. Just after a test vector was generated for a target fault, X inputs may remain. But the X inputs are specified by random fill or static/dynamic test compaction [6] because faults other than the target fault may be detected by fault simulation. Thus, no Xs at the primary inputs remain in the final test vectors.

However, some of primary input values may be changed to opposite logic values without losing fault coverage. We can regard such input values as *don't care* (X). In this paper, we propose a method for identifying X inputs of test vectors in a given test set. Since there are many combinations of X inputs in general, the proposed method finds one including X inputs as many as possible, by using fault simulation and procedures similar to implication and justification of ATPG algorithms. The implication and justification procedures are restricted so that the obtained test set covers the original test set. Experimental results for ISCAS benchmark circuits show that approximately 66% of inputs of un-compacted test sets are Xs in average. Even if compacted test sets [8] were given, the method found approximately 47 % of inputs were Xs.

After a test set including X inputs was obtained, arbitrary logic values can be assigned to the X inputs. Hence we can make the test set to have desirable features without losing fault coverage and without adding new test vectors. We also discuss in this paper how logic values are reassigned to the X inputs where several applications exist.

This paper is organized as follows. In Section 2, we give problem formulation and definitions of terminology used in this paper, and then show overview of the proposed method. In Section 3, we present more details of the proposed method. In Section 4, experimental results for benchmark circuits are given. In Section 5, we present several applications of the proposed method, and we conclude this paper in Section 6.

2. Preliminary

2.1 Problem formulation

In this paper, we treat test sets generated for single stuck-at faults of combinational circuits or full-scan sequential circuits. Given a circuit and its test set T in which every primary input value of test vectors has been specified to either 0 or 1, we compute test set T' including some Xs (*don't cares*), where test set T' has the following properties:

(1) T' covers T.
(2) T' contains Xs as many as possible.
(3) Fault coverage of T' is equal to that of T.

We show a simple example. Suppose that test set T in Table 1(a) was generated for a circuit in Fig. 1. Test set T' in Table 1(b) is one of the solutions. Test vector t_1 detects fault $a/0$, $b/0$, and $c/1$, where s/v denotes stuck-at v fault on signal line s. While $a/0$ have to be detected by t_1, fault $c/1$ does not have to be detected by t_1 because t_3 detects it too. Hence value 0 at input c becomes an X. Similarly, value 0 at input a of t_4 becomes an X. Thus, test set T' in Table 1(b) is obtained.

In this paper, a given test set and a derived test set are denoted by T and T', respectively. Similarly, a test vector in T and a test vector in T' are denoted by t_i and t_i', respectively. Note that t_i' is derived from t_i, as shown in Table 1.

2.2 Definitions

We define some terminology used in this paper below. Given a test set T, if any fault detected by test vector t_i in T is detected by at least one test vector in T-$\{t_i\}$, t_i is called a *redundant* test vector. When t_i is redundant, fault coverage of T-$\{t_i\}$ keeps that of T. If fault f is detected by t_j in T, but not detected by any test vector in T-$\{t_j\}$, f is called an *essential fault of* t_j [7]. Redundant test vectors do not detect essential faults. If there is no redundant test vector in T, T is called a *minimal* test set. A minimal test set can be obtained easily by double detection [8].

2.3 Overview of the proposed method

We explain a basic procedure to obtain test set T' from given test set T. In order that T' might cover T and that fault coverage of T' might keep that of T, essential faults of t_i in T must be detected by t_i' in T'. Not-essential faults of t_i, however, don't have to be detected by t_i' because they have a chance to be detected by another test vector. Hence we first fix logic values to detect essential faults of each test vector, and then fix logic values to detect the other faults that are undetected by the fixed values for the essential faults. Unfixed logic values are dealt with as Xs after all.

In Fig. 2, we give a basic procedure to obtain T' from T. For each test vector t_i, we collect essential faults at Step 1. Then, calculate logic values of t_i which contribute to the detection of the essential faults, and let the value at primary inputs be t_i' at Step 2. The details of this calculation will be described in Section 3. Note that t_i' at Step 2 is still intermediate, not final one yet. As t_i' may detect faults other than the essential faults, we perform fault simulation for t_i' at Step 3. The fault simulator used in this procedure can deal with test vectors including X inputs. By applying these three steps for each test vector, test set T', which is still temporary, is obtained.

Since some faults are not detected by the temporary T', we return some of the Xs in t_i' to the original value of t_i, such that all faults are detected. At Step 4, we collect faults which are undetected but detectable by T, and let a set of the collected faults be G. Then, calculate logic values of t_i which contribute to the detection of faults in G at Step 5, and add the value at primary inputs to t_i'.

3. Logic values to detect a fault

As described above, the proposed method calculates and fixes necessary logic values of each test vector, and regards the unfixed logic values as Xs. In this section we describes how to calculate the logic values to be fixed.

3.1 Selection of fault propagation paths

When a test vector detects a fault, there exist internal logic values to activate the fault and to sensitize at least one fault

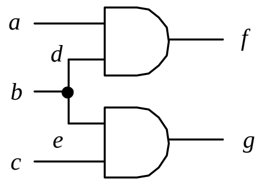


Fig. 1: Example circuit

Table 1(a): Given test set T

	a b c
t_1	1 1 0
t_2	1 0 1
t_3	0 1 0
t_4	0 1 1

Table 1(b): Obtained test set T'

	a b c
t_1'	1 1 x
t_2'	1 0 1
t_3'	0 1 0
t_4'	x 1 1

```
Basic Procedure X-search(C, T)
     Circuit C; Test set T;
{
   for each test pattern t_i in T {
     F=collect_essential_fault(t_i);              /* step 1 */
     t_i' = find_value(F);                        /* step 2 */
     fault_simulation(t_i');                      /* step 3 */
   }
   for each test pattern t_i in T {
     G=collect_undetected_fault(t_i);             /* step 4 */
     t_i' += find_value(G);                       /* step 5 */
     fault_simulation(t_i');                      /* step 6 */
   }
   return T' composed of t_i';
}
```

Fig. 2: Basic procedure for identifying X inputs

propagation path to a primary output. We collect such internal logic values, which are a value on the faulty line and values on fan-in lines of gates along the fault propagation path. Then, we calculate logic values at primary inputs to satisfy the values by implication and justification procedures used in ATGP algorithms.

Sometimes more than one fault propagation path exists. In order to leave Xs at as many inputs as possible, we select only one among them, because sensitization of one of the paths is enough to detect the fault. Since values to be fixed are different depending on the path selected, it is important which path is selected. We select a path to a primary output to which more faults are propagated. This is because, if a value is related to sensitization of many paths, total number of the fixed values becomes small potentially.

3.2 Limited implication and limited justification

In order to fix values at primary inputs to activate a fault and to sensitize a fault propagation path, we employ procedures similar to implication and justification used in ATGP algorithms. Unlike procedures used in ATPG algorithms, a limitation exists so that conflicts with the original value must be avoided. We refer to the implication and the justification used in the proposed method as *limited implication* and *limited justification*.

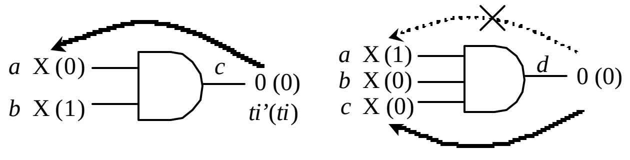
Fig. 3(a) illustrates an example of limited implication. Logic values outside the parenthesis mean values for t_i', and logic values inside the parenthesis mean values for t_i. Suppose that values of gate inputs, *a* and *b*, have not been assigned yet for t_i' and logic value of the gate output *c* is newly assigned to 0. In this case, the limited implication procedure derives "*c*=0 implies *a*=0" because *b* cannot take value 0 for t_i' due to *b*=1 for t_i.

Fig. 3(b) illustrates an example of limited justification. Consider to justify logic value 0 on output *d* when values of gate inputs, *a*, *b*, and *c*, have not been assigned yet for t_i'. In this case, the limited justification procedure can select "*b*=0" or "*c*=0", but cannot select "*a*=0" because value of *a* was 1 for t_i.

By applying limited implication and limited justification until no unjustified line remains, primary input values to satisfy value assignments to detect the faults are found. If no value was assigned to a primary input, the primary input takes X in t_i'. Otherwise, the value, which is the same of t_i, is fixed in t_i'. Note that backtracking is never needed during the procedures because the original test guarantees that unjustified lines are justifiable.

3.3 Extended implication

Limited implication and limited justification are based on 3-valued logic (0, 1, X). Hence it takes only fault-free values into account, and ignores faulty values. Due to the 3-valued logic in



(a) Limited implication (b) Limited justification

Fig. 3: Limited implication and justification

limited implication and limited justification, the obtained test set *T'* may miss some detectable faults, even though the faults has been treated explicitly. We show an example in Fig. 4. Test vector (*a*, *b*, *c*) = (0,0,0) detects stuck-at 1 fault on line *b*. Since a fault propagation path is *b*-*e*-*f* as shown in Fig. 4(a), *b*=0, *d*=0 and *c*=0 are required to detect the fault. Since *b* and *c* are primary inputs, and *d*=0 is implied by *b*=0, we don't need additional assignments. As a result, value of *a* becomes an X as shown in Fig. 4(b). However, as shown in Fig. 4(c), detection of stuck-at 1 fault on *b* is not guaranteed, i.e., it is not detected when *a*=1. Such a case seldom occurs, but we give an additional procedure to solve the problem.

The reason why such undetected faults are produced is that limited implication and limited justification do not look at values of the faulty circuit. For example in Fig. 4, while *d*=0 is required for the faulty circuit, *d*=0 is implied by *b*=0 on the fault-free circuit. This implication, however, results in *d*=1 on the faulty circuit. In order to avoid such undetected faults, we propose *extended implication* that is applied to backward implication.

In the extended implication, when a fault effect is propa-

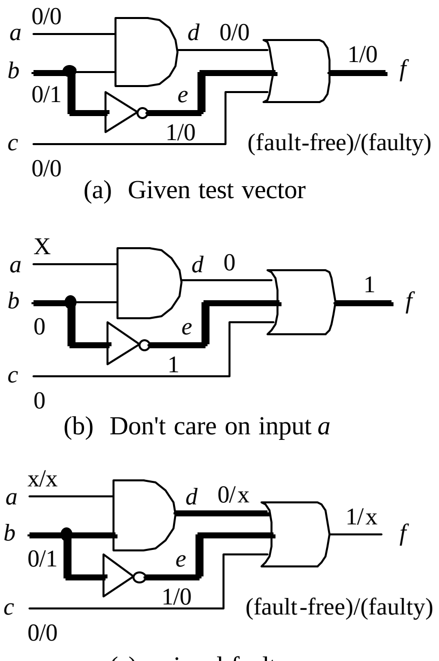


Fig. 4: Missed fault by limited implication

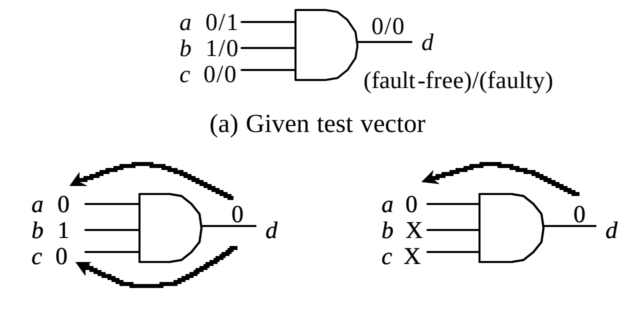


Fig. 5: Extended implication

gated to at least one fan-in line of a gate, values of all fan-in lines of the gate are fixed. An example of extended implication is given in Fig. 5. Assume that values of gate inputs, *a*, *b*, *c* were 0, 1, 0 for t_i, respectively, and faults were propagated to *a* and *b*, as shown in Fig. 5(a). Now, consider a case that values of gate inputs, *a*, *b*, *c* have not been assigned yet for t_i' and logic value of gate output *d* is newly assigned to 0. Extended implication is "*d*=0 implies *a*=0, *b*=1, and *c*=0" as shown in Fig. 5(b). Limited implication described in Section 3.2 cannot fix any input value, and limited justification take either *a*=0 or *c*=0, as shown in Fig. 5(c). On the other hand, extended implication may fix unnecessary values to detect a fault. For example of Fig 5(b), *c*=0 is enough to *d*=0. So extended implication should not be used for many faults. Note that when extended implication is used, forward implication from the fault site are not performed to avoid missing cases that need extended implication.

Fig. 6 shows the complete procedure of the proposed method to identify X inputs in a given test set. Steps 7 to Step 9 are added to the basic procedure in Fig. 2. The extended implication is used only at Step 8.

4. Experimental results

We implemented the proposed method on a PC (PentiumIII 700MHz, 384MB memory) using C programming language and applied it for ISCAS'85 and full-scan version of ISCAS'89 benchmark circuits. We gave two kinds of test sets for each circuit, an un-compacted test set and a compacted one [8]. Table 2 and Table 3 show results for un-compacted test sets and compacted test sets, respectively.

The first three columns of the Tables give the circuit name, the number of primary inputs, and the number of test vectors of the given test set. The next three columns show the average percentage of X inputs for each test vector, the percentage of X inputs of the test vector with the most Xs, and the percentage of X inputs of the test vector with the least Xs. The next two columns show the number of target faults, and the number of faults

```
Complete Procedure X-search(C, T)
    Circuit C; Test set T;
{
  for each test pattern t_i in T {
    F=collect_essential_fault(t_i);            /* step 1 */
    t_i' = find_value(F);                      /* step 2 */
    fault_simulation(t_i');                    /* step 3 */
  }
  for each test pattern t_i in T {
    G=collect_undetected_fault(t_i);           /* step 4 */
    t_i' += find_value(G);                     /* step 5 */
    fault_simulation(t_i');                    /* step 6 */
  }
  for each test pattern t_i in T {
    H=collect_undetected_fault(t_i);           /* step 7 */
    t_i' += extended_find_value(H);            /* step 8 */
    fault_simulation(t_i');                    /* step 9 */
  }
  return T' composed of t_i';
}
```

Fig. 6: Complete procedure for identifying X inputs

which were undetected after the basic procedure. All the undetected faults were finally detected using the procedure with extended implication described in Section 3.3. The last column shows CPU time in second.

For un-compacted test sets, the proposed method found that values of approximately 66 % of primary inputs are X in average. Even for compacted test sets, values of approximately 47% of primary inputs were X. However, the percentage of X inputs depends on the circuit and its test set. In general, The more the number of primary inputs is, the more X inputs its test set has. When the given test set is not minimal, there exists a test vector of which all inputs are Xs. Faults detected by using extended implication are at most 2 % for every circuit. CPU time strongly depends on the number of test vectors of the given test set and circuit size. This is because the proposed method processes each test vector in serial.

5. Applications

Since we can re-assign arbitrary logic value to X inputs identified by the proposed method, we can make a test set to have desirable features without losing fault coverage and without increasing the number of test vectors. In this section, we present some applications of the proposed method.

5.1 Test compaction

The most typical application is test compaction. For example, test set *T'* in Table 1(b) can be compacted by merging t_1' with t_4'. This is static compaction [6] and would be useful when the size of the given test set is large. Even if it is impossible to compact *T'* by static compaction, we can apply dynamic compaction for *T'* so as to make some test vectors be redundant [9].

Reduction of test application time for full-scan circuits is also possible by *T'*. If there exist Xs at pseudo-primary inputs close to the scan-out output, the scan-in operation can be omitted. Similarly, if there exist Xs at pseudo-primary outputs close to the scan-in input, the scan-out operation can be omitted. Sub-

Table 2: Results for un-compacted test sets

circuit	#PIs	#tests	%X-ave	%X-max	%X-min	#faults	#flt-ext	time(sec)
c432	36	54	49.0	100	13.9	520	3	0.03
c499	41	94	20.5	100	0	750	41	0.05
c880	60	78	64.3	100	11.7	942	7	0.07
c1355	41	129	25.3	100	0	1566	0	0.16
c1908	33	150	33.7	100	0.0	1862	17	0.32
c2670	233	142	84.0	100	27.9	2630	17	0.69
c3540	50	207	59.2	100	12.0	3291	42	1.71
c5315	178	186	80.7	100	25.8	5291	34	2.04
c6288	32	38	13.5	100	0	7710	16	1.91
c7552	207	290	76.2	100	16.4	7419	90	5.01
s1238	32	195	62.0	100	28.1	1286	1	0.16
s1423	91	93	76.5	100	20.9	1501	29	0.12
s1494	14	166	38.1	100	7.1	1494	1	0.16
s5378	214	333	88.7	100	19.2	4563	11	2.89
s9234	247	480	88.3	100	23.9	6475	58	7.74
s13207	700	586	96.0	100	33.0	9664	158	14.84
s15850	611	500	94.0	100	24.7	11336	77	15.71
s35932	1763	76	84.8	99.9	0.6	35110	16	18.44
s38417	1664	1243	96.7	100	24.2	31015	271	108.53
s38584	1464	854	96.3	100	8.3	34797	187	75.02
Average			66.3	99.9	14.8			

Table 3: Results for compacted test sets

circuit	#PIs	#tests	%X-ave	%X-max	%X-min	#faults	#flt-ext	time(sec)
c432	36	28	46.9	66.7	8.3	520	0	0.02
c499	41	52	0.6	14.6	0.0	750	34	0.03
c880	60	21	31.7	63.3	11.7	942	0	0.04
c1355	41	84	0.0	2.4	0.0	1566	2	0.13
c1908	33	106	15.8	63.6	3.0	1862	22	0.28
c2670	233	45	70.1	86.3	20.6	2630	3	0.33
c3540	50	93	49.3	76.0	26.0	3291	58	0.98
c5315	178	186	59.5	89.3	23.0	5291	3	0.81
c6288	32	14	0	0	0	7710	0	1.18
c7552	207	75	52.7	82.1	10.6	7419	64	1.90
s1238	32	125	55.0	65.6	25.0	1286	0	0.10
s1423	91	24	41.1	71.4	17.6	1501	7	0.08
s1494	14	100	25.5	57.1	0.0	1494	0	0.11
s5378	214	100	71.0	88.3	8.9	4563	5	1.13
s9234	247	111	67.2	87.5	36.0	6475	35	2.45
s13207	700	235	91.6	98.4	16.4	9664	149	7.34
s15850	611	97	76.1	96.6	24.2	11336	60	4.87
s35932	1763	12	34.4	83.5	0.0	35110	345	9.03
s38417	1664	87	73.4	89.4	20.1	31015	93	13.73
s38584	1464	114	79.7	95.6	14.3	34797	88	15.78
Average			47.0	68.8	13.2			

sequently, we can reduce scan shift operation, which contributes to the reduction of test application time [10],[11].

Another application is test compression using encoding technique. For examople, the storage of a test set on a ROM is reduced by statistically encoding the test set [12]. The test set including Xs would bring smaller codes.

5.2 Reduction of power dissipation during testing

Power dissipation under testing condition is sometimes 100-200% higher than that of normal operation [13]. Therefore fault-free circuits may produce faulty responses for some test vectors even though each test vector has statically no problem for the circuits. Since power dissipation of LSIs increases in proportional to switching activity, it can decrease by assigning appropriate logic values to Xs [14].

5.3 Enhancement of test quality

It is known that defect coverage of a test set is improved if the test set is generated so that each stuck-at fault is detected more than once [2][15]. By using X inputs identified by our method, the test set can detect each faults more times than the original one, and would improve defect coverage.

The method proposed in this paper has not developed to be applied to a test set for path delay faults. But it is possible to identify Xs of the test set for path delay faults, because internal logic values to sensitize target paths can be specified easily. Since power supply noise affects propagation delay of paths [16], by using Xs, the given test set can be modified to one with the worst case delay.

6. Conclusions

We proposed a method for identifying X (don’t care) inputs of test vectors in a given test set. The proposed method found by using fault simulation and procedures similar to implication and justification of an ATPG algorithm. Experimental results for ISCAS benchmark circuits showed that approximately 66% of inputs of un-compacted test sets could be X in average. Even if compact test sets were given, the method found approximately 47 % of inputs were X although the non-compacted test sets have more X inputs than the compacted ones. We also discussed how logic values are reassigned to the X inputs and presented several applications. We will report the effectiveness of the test pattern modification in the future.

References

[1] M. L. Bushnell, V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*, Kluwer Academic Publishers, 2000.

[2] S. C. Ma, P. Franco, and E. J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experiment Results," 1995 International Test Conf., pp. 663-672, Oct. 1995.

[3] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," 1991 International Test Conf., pp. 194-203, Oct. 1991.

[4] S. Wang, S. K. Gupta, “ATPG for Heat Dissipation Minimization during Test Application,” IEEE Trans. Computer, Vol. 47, No. 2, pp. 256-262, Feb. 1998.

[5] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," IEEE Trans. on CAD., pp. 126-137, Jan. 1988.

[6] P. Goel, and B. C. Rosales, "Test Generation and Dynamic Compaction of Tests," Digest of Papers 1979 Test Conf., pp. 189-192, Oct. 1979.

[7] J. -S. Chang, C. -S. Lin, "Test Set Compaction for Combinational Circuits," First Asian Test Symposium, pp. 20-25, Nov. 1992.

[8] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 12, pp.1496-1504, Dec. 1995.

[9] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "ROTCO: A Reverse Order Test Compaction Technique," 1992 IEEE EURO-ASIC Conference, pp. 189-194, Sept. 1992.

[10] S. Y. Lee and K. K. Saluja, "An Algorithm to Reduce Test Application Time in Full Scan Designs" Proc. Int'l Conf. on CAD, pp. 17-20, Nov. 1992.

[11] Y. Higami, S. Kajihara, and K. Kinoshita, "Reduced Scan Shift: A New Testing Method for Sequential Circuits," IEEE International Test Conference, pp. 624-630, Oct. 1994.

[12] V. Iyengar, K. Chakrabarty, and B. T. Murray, "Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets," 16th VLSI Test Symposium, pp. 418-423, 1998.

[13] Y. Zorian, “A Distributed BIST Control Scheme for Complex VLSI Devices,” 11th VLSI Test Symposium, pp. 4-9, 1993.

[14] R. Sankaralingam, R. R. Oruganti, N. A. Touba, “Static Compaction Techniques to Control Scan Vector Power Dissipation,” 18th VLSI Test Symposium, pp. 35-40, 2000.

[15] S. M. Reddy, I. Pomeranz, and S. Kajihara, "Compact Test Sets for High Defect Coverage," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 8, pp.923-930, Aug. 1997.

[16] A. Krstic, Y. -M. Jiang, K. -T. Cheng, "Delay Testing Considering Power Supply Noise Effects," International Test Conf., pp. 181-190, Sept. 1999.