Novel Techniques for Improving Testability Analysis

Yin-He Su, Ching-Hwa Cheng, and Shih-Chieh Chang

Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan, Republic of China
{yhs87, chc, scchang @ cs.ccu.edu.tw }

Abstract

The purpose of a testability analysis program is to estimate the difficulty of testing a fault. A good measurement can give an early warning about the testing problem so as to provide guidance in improving the testability of a circuit. There have been researches attempting to efficiently compute the testability analysis. Among those, the Controllability and Observability Procedure COP [1] can calculate the testability value of a stuck-at fault efficiently in a tree-structured circuit but may be very inaccurate for a general circuit. The inaccuracy in COP is due to the ignorance of signal correlations. Recently, the algorithm of TAIR in [2] proposes a testability analysis algorithm, which starts from the result of COP and then gradually improves the result by applying a set of rules. The set of rules in TAIR can capture some signal correlations and therefore the results of TAIR are more accurate than COP. In this paper, we first prove that the rules in TAIR can be replaced by a closed-form formulation. Then, based on the closed-form formulation, we proposed two novel techniques to further improve the testability analysis results. Our experimental results have shown improvement over the results of TAIR.

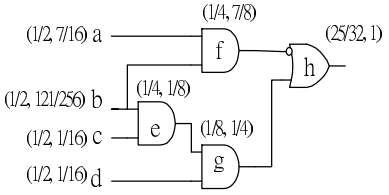
1 Introduction

Built-in Self-Test (BIST) that applies test patterns from on chip hardware is becoming attractive because it enables at speed testing. The most common way to provide test patterns in BIST is to construct a Linear Feedback Shift Register (LFSR) where pseudo random patterns can be generated. To determine the fault coverage from a set of random patterns generated from LFSR, one can perform fault simulation on the set of patterns. However, the number of random patterns can be large and the time for fault simulation can be prohibited long. Another alternative is to estimate the fault coverage from the testability analysis, which describes the difficulties of testing the faults. On the other hand, the problem of finding the exact testability analysis of a fault is an NP complete problem and normally it is a tradeoff between accuracy and CPU time. If the fault coverage is insufficient, test point insertion can be adopted to improve the fault coverage requirement. Several test point insertion algorithms [3] also rely on the testability analysis to determine where observability points or controlling points are added. As a result, a good testability measurement is important in designing a circuit with BIST because it can provide estimation for the fault coverage and also can give good guidance to improve the random testability of a circuit.

There have been researches in the area of computing the detection probabilities or testability analysis [4] of a fault assuming random patterns are applied to the circuit under test. Let the controllability of a node be the probability of the node evaluated to 1 and the observability of a node to be the probability of the node observed at a primary output. The algorithm **COP** (Controllability / Observability Procedure) [1] computes the observability and controllability of each node by two sweeps of the circuit. During the computation, all signals are assumed to be independent. For example in Fig. 1, let the 1'controllabilities of all inputs be 1/2. The algorithm COP estimates the *1*-controllability of node *e* to be *1/4* and the observability to be *1/8*. The testability for wire $e \rightarrow g$ stuck-at-1 fault is calculated to be *{1-(1/4)}*1/8 = 3/32*. Although COP can quickly compute the detection probabilities of all faults, the probabilities computed by COP are generally inaccurate due to the ignorance of signal correlation. In the table of Fig.1, the first column shows a wire stuck at fault and the second (third) column shows the exact (COP) detection probability for the corresponding stuck at fault. The exact results are obtained by some exhaustive methods. As can be seen from the table, none of estimations by COP are the same as the exact value in this example.

Recently, the algorithm **TAIR** [2] proposes to gradually improve the COP results by taking into account some signal correlations. Consider to determine the testability analysis of a stuck at fault. To activate and to observe the

fault, certain value assignments must be set for the fault, the assignments of which can also imply new value assignments. The process of obtaining new value assignments is called the *implication* process. TAIR observes that some signal correlations can be modeled by applying rules during the implication process. When the condition of a rule is met, TAIR multiplies the COP result by some factor. Their experimental results have shown that the new estimations are more accurate than COP. Consider again to find the detection probability for $e \rightarrow g$ stuck-at-1 fault in Fig. 1. To activate the fault, node *e* must be assigned *0* and also, to propagate the fault, node *d* and node *f* must be assigned *1*. Since node *f* is *1*, node *a* and node *b* must both be *1*. In the algorithm, TAIR can recognize that two signal assignments *{b=1, e=0}* are correlated and then multiply a factor of *2/3* to the COP result (3/32) to get the exact result of *1/16*. The testability analysis results of TAIR for the circuit in Fig. 1 are shown in the fourth column in the table where the boldface means the TAIR's results are the same as the exact results.



(1'controllability, observability)

Testability			
Wire	Exact	COP	TAIR
f→h (s-a-1)	11/16	21/32	21/32
g→h (s-a-0)	1/16	1/32	**1/16**
b→e (s-a-0)	1/16	1/32	**1/16**
b→e (s-a-1)	0	1/32	**0**
c→e (s-a-1)	1/16	1/32	**1/16**
a→f (s-a-1)	3/16	7/32	7/32
b→f (s-a-0)	1/4	7/32	7/32
e→g (s-a-1)	1/16	3/32	**1/16**

Fig. 1 An example of testability analysis from exact, COP, and TAIR method.

As can be seen from the table, there are still quite a number of TAIR testability results different from the exact values. The purposes of this paper are two fold. First, in TAIR, it recursively applies a set of rules to improve the COP results. We show that the algorithm of applying rules can be replaced by a closed-form formulation. Secondly, based on the closed-form formulation, we further improve the TAIR results by two novel techniques. Basically, these two techniques can recognize some "far away" signal correlations compared to TAIR and as a result can obtain more accurate estimation.

2 Testability Analysis and Its Relation with Mandatory Assignments

In this section, we quickly review the algorithms of COP and TAIR. To simplify the discussion, let node n_i be a node in the circuit and v_i be *0* or *1*. First, let us have some definitions.

Definition 1: The *signal probability* or *1'controllability* of a node *n*, $p(n=v)$, is the probability to evaluate node *n* to *v*, and the *multiple signal probability* $p(n_0=v_0, n_1=v_1, \ldots, n_k=v_k)$ is the probability to simultaneously evaluate each node n_i to v_i.

Note that, if all signal assignments $\{n_0=v_0, n_1=v_1, \ldots, n_k=v_k\}$ are structurally independent, we have

$$p(n_0=v_0, n_1=v_1, \ldots, n_k=v_k) = p(n_0=v_0)*p(n_1=v_1)* \ldots *p(n_k=v_k).$$

In a tree-structured circuit, because signal assignments for the immediate fanins of a node are all independent, the signal probability of the node can exactly computed by multiplying the signal probabilities of the node's fanins. The observability of a node can also be exactly computed by some rules. Based on the assumption of signal independencies, the COP algorithm estimates the controllability and the observability of a node using the same tree formula. Let the COP observability of a node *n* be represented by *obv(n)*. The observability of node *e* shown in Fig. 1 is recursively computed by the following formula:

$$obv(e) = p(d=1)*obv(g) = p(d=1)*p(f=1)*obv(h) = p(d=1)*p(f=1).$$

where *obv(h)* is assigned *1* because node *h* is a primary output. Since $obv(e) = obv(e \rightarrow g)$, the COP testability for fault $e \rightarrow g$ s-a-1 is equal to $obv(e \rightarrow g)*p(e=0)$. Therefore, the COP testability for $e \rightarrow g$ s-a-1 is

$$obv(e)*p(e=0) = p(d=1)*p(f=1)*p(e=0).$$

After briefly describing the COP algorithm, we now review the TAIR algorithm, which uses the implication process to improve the testability analysis. A node is said to be a *dominator* of a wire (fault) if all paths from the wire (fault) to any primary output, must pass through this node. Also, the *side inputs* of a dominator of a wire (fault) are those direct inputs of the dominator not in the transitive fanout of the wire (fault). During a stuck at fault test, certain value assignments must be set to activate/observe the fault. These value assignments are called the *mandatory assignments* (MAs) for the stuck-at fault test.

Definition 2: The *root* MAs of a wire stuck-at fault are the MAs assigned to the side inputs of a dominator (sensitization) and the MA assigned to the source of the wire (activation).

For example, in Fig. 1, the root MAs for wire $c \rightarrow e$ stuck-at-1 fault are *{b=1, d=1, f=1, c=0}* where MAs *{b=1, d=1, f=1}* are to sensitize the fault and MA *{c=0}* is to activate the fault. The implied MA *{a=1}*, however, is not a root MA.

Lemma 1: From [2], for a stuck-at fault *f*, let MAs $\{n_0=v_0, n_1=v_1, \ldots, n_k=v_k\}$ be the root MAs. The testability analysis computed by COP for *f* is

$$Const * p(n_0 = v_0) * p(n_1 = v_1) * \ldots * p(n_k = v_k) = Const \times \prod_{(n_i = v_i) \in \text{Root MAs}} p(n_i = v_i)$$

Given the example in Fig. 1, the COP testability for $e \rightarrow g$ s-a-1 fault is determined as $p(d=1)*p(f=1)*p(e=0)$. We emphasize again that MAs $\{d=1, f=1, e=0\}$ are exactly the root MAs for $e \rightarrow g$ s-a-1 fault. For Lemma 1, COP computes the testability by multiplying the signal probability of each root MA with some constant, which come from the formula for multiple fanouts by COP. However, the signal assignments for the root MAs are not independent because all test vectors must simultaneously satisfy the root MA conditions. Therefore, a more accurate estimation for testability analysis should compute the multiple signal probability $p(n_0=v_0, n_1=v_1, \ldots, n_k=v_k)$, instead of computing $p(n_0=v_0)*p(n_1=v_1)* \ldots *p(n_k=v_k)$ by assuming signal independencies of root MAs. For example in Fig. 1, COP gives a wrong testability of *3/32* for $e \rightarrow g$ s-a-1 fault from $p(d=1)*p(f=1)*p(e=0)$. The exact result can be obtained from the multiple signal probability $p(d=1, f=1, e=0)=1/16$.

In the algorithm of TAIR, it assumes that the controllability and the observability for each node are derived a priori by COP. For a stuck-at fault, TAIR first assigns the root MAs for the fault. Then, new additional MAs are implied from forward and backward propagating root MAs. Using the information from these new implied MAs, it derives a set of rules in Table 1 to model signal correlations. When the condition of a rule is met, the COP testability is multiplied by the corresponding correlation factor.

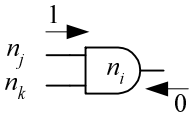
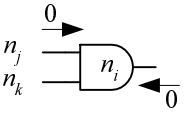
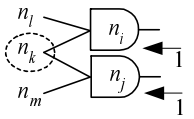
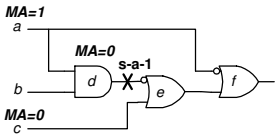
Rule #	The condition of a rule	Correlation Factor (*CF*)
Rule 1		$CF1 = p(n_k=0)/p(n_i=0)$
Rule 2		$CF2 = 1/p(n_i=0)$
Rule 3		$CF3 = 1/p(n_k=1)$

Table 1 The condition and correlation factor of each rule.

Consider the example, $d \rightarrow e$ s-a-1 fault in Fig. 2. The COP testability of the fault is determined as $p(c=0)*p(a=1)*p(d=0)$ where $\{c=0, a=1, d=0\}$ are root MAs. This testability is incorrect because MAs $\{a=1, d=0\}$ are dependent. One can check that MAs $\{a=1, d=0\}$ leads to MA $\{b=0\}$. As a result, the multiple signal probability $p(c=0, a=1, d=0)$ should be equal to $p(c=0, a=1, b=0)$. Suppose the new signal assignments $\{c=0, a=1, b=0\}$ are independent. It is easy to see that

$p(c=0, a=1, b=0) = p(c=0) * p(a=1) * p(b=0)$
$= p(c=0) * p(a=1) * \{p(b=0)/p(d=0)\} * p(d=0)$
$= \{COP\ testability\ result\} * \{p(b=0)/p(d=0)\}$.



Node	1'Controlability	Observability
a	1/2	57/64
b	1/2	1/8
c	1/2	1/8
d	1/4	1/4
e	7/8	1/2
f	15/16	1

Fig. 2 The relation between COP testability and MAs for a stuck-at fault.

Consequently, to correct the COP testability, TAIR multiplies the old COP result, *3/16* by $p(b=0)/p(d=0) = 2/3$, the correlation factor. Thus, TAIR obtains the exact value *1/8*. Note that assignments $\{c=0, a=1, b=0\}$ are independent since nodes *a*, *b*, and *c* are all primary inputs. If they are not independent, TAIR may recursively apply rules to obtain more accurate results.

The entire TAIR algorithm is shown in Fig. 3, assume COP is done before this routine. Given the COP results, TAIR first computes the root MAs for a target stuck-at fault. Then, they perform direct implication on these MAs. During the implication process, if the MAs cannot be consistently justified, TAIR returns zero for the fault. Otherwise, when the condition of a rule is met, the corresponding correlation factor is applied. The procedure for each target fault continues until no rule can be met. Following the above process, TAIR is able to enhance the COP testability for each fault under consideration. In addition, the testabilities of many redundant faults can be naturally set to zero.

```
Improve_testability_analysis_for_a_fault(f_i)
{
Push_rootMAs_into_stack(f_i)
while (stack_is_not_empty)      {
        (n_i=v_i) = pop(stack)
        imply(n_i=v_i)
        If (MA_is_inconsistent) /* the fault is redundant */
                return 0
        switch (condition of MAs) {
                case rule1 : CF = CF * CF1
                case rule2 : CF = CF * CF2
                case rule3 : CF = CF * CF3 }
        }
        return CF
}
```

Fig. 3 The rule-based TAIR algorithm.

3 A Closed-Form Formulation for TAIR

The algorithm of TAIR recursively applies a set of rules to gradually improve the testability analysis for a fault. However, depending on the order of processing MAs, the rules and the corresponding correlation factors applied for improving COP results, can be different. In this section, we show that the set of rules in Table 1 can be replaced by a closed-form formulation, from which we can prove that the results of TAIR are independent to the order of processing MAs. In addition, with this closed-form formulation, we propose two new techniques to further improve the results of TAIR. Without losing the generality, let us assume that the circuit under consideration consists only two input AND/OR/INVERTER gates. One can easily extend to gates with multiple inputs or other types of gates such as an XOR gate.

Definition 3: An MA $\{n=v_i\}$ is said to be a *frontier* MA if
1. node n is a primary input or
2. all of n's inputs do not have MAs assigned.

For example, consider the $a \rightarrow d$ s-a-1 fault in Fig. 4. The root MAs are $\{a=0, b=1, f=0, o=0\}$. After performing direct implications, we have MAs = $\{a=0, b=1, f=0, o=0, j=0, i=0, m=0\}$. Among them, MAs $\{a=0, b=1, j=0, m=0\}$ are the frontier MAs because nodes *a*, *b*, and *m* are primary inputs, while the inputs of node *j* do not have MAs assigned. Note that the MAs $\{o=0, f=0, i=0\}$ are not frontier MAs.

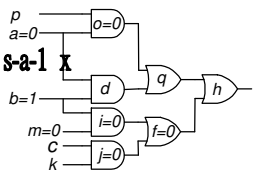


Fig. 4 The frontier MAs.

Theorem 1: The product of probabilities for root MAs is modified to that of probabilities for frontier MAs by applying the correlation factors in TAIR.

Lemma 2: The final correlation factor ***CF*** in Fig. 3 is equal to the following equation.

$$CF = \frac{\prod_{(n_i=v_i)\in \text{Frontier MAs}} p(n_i = v_i)}{\prod_{(n_i=v_i)\in \text{Root MAs}} p(n_i = v_i)}$$

Lemma 3: The final ***CF*** is independence of the order in processing MAs and of the order in implementing the implication rules.

For example in Fig. 4, suppose we start with the

$$\text{COP result} = \prod_{(n_i=v_i)\in \text{Root MAs}} p(n_i = v_i)$$

$= p(a=0)*p(b=1)*p(f=0)*p(o=0)$ where $\{a=0, b=1, f=0, o=0\}$ are the root MAs. After performing the direct implication on these root MAs, we can apply two rules. Rule 2 is applied to node ***o*** because of the MAs $\{o=0, a=0\}$ and Rule 1 is applied to node *i* because of the MAs $\{b=1, i=0\}$. The final correlation factors $CF = CF1 * CF2 = \{p(m=0)/p(i=0)\} * \{1/p(o=0)\}$ is obtained by the algorithm given in Fig. 3. Using the formula in Lemma 2, we can first find the frontier MAs which are $\{a=0, b=1, j=0, m=0\}$. The final correlation factor is therefore determined as

$$CF = \frac{\prod_{(n_i=v_i)\in \text{Frontier MAs}} p(n_i = v_i)}{\prod_{(n_i=v_i)\in \text{Root MAs}} p(n_i = v_i)} = \frac{p(a=0)\times p(b=1)\times p(j=0)\times p(m=0)}{p(a=0)\times p(b=1)\times p(f=0)\times p(o=0)}$$

Because $p(f=0) = p(j=0)*p(i=0)$, substituting this into the above equation, we can have *CF* equal to $p(m=0)/\{p(o=0)*p(i=0)\}$. It can be easily checked that the result of *CF* derived from the algorithm of Fig. 3 is the same as the result obtained from Lemma 3.

4 Novel Techniques for Testability Improvement

The TAIR algorithm recognizes some correlation when one MA "runs into" another MA on some node. In other words, TAIR can model some signal correlations that are topologically "close" to each other. However, from the table in Fig. 1, there exist some correlations that are not identified by TAIR. In this section, we propose two novel techniques to improve the results. The first one termed as the transitive fanout technique attempts to taken into account signal correlation between a node and nodes in its transitive fanouts. The other technique termed as the MA insertion technique re-computes the testability analysis by inserting MAs to some nodes.

4.1 Transitive fanout technique

In the transitive fanout technique, we attempt to model the correlation between a node and another node in its transitive fanout. We illustrate how the improvement can be achieved by the example in Fig. 5. Let us try to compute the testability analysis for $a \rightarrow f$ *s-a-1*. It can be shown that the exact testability should compute the multiple signal probability $p(b=1, g=0, a=0)$. Since MA $\{a=0\}$ is independent to others so it can be written to be $p(a=0)*p(b=1, g=0)$. Because MA $b=1$ and MA $g=0$ are topologically far away, there are no rules matched in TAIR and no correlation can be found. However, we can re-write $p(b=1, g=0)$ by the followings.
$p(b=1, g=0) = p(b=1)*p(g=0|b=1)$.

The conditional probability of $p(g=0|b=1)$ can be computed as follows. We first assign $b=1$ and then re-compute the probability of $g=0$. In this example, under the condition of $b=1$, the conditional probability for $g=0$ is equal to 3/4. Therefore, we have

$p(b=1, g=0)=p(b=1)*p(g=0|b=1) = 1/2*3/4 = 3/8.$

And the final testability is equal to $p(a=0, b=1, g=0) = p(a=0)*p(b=1, g=0) = 1/2 *3/8 = 3/16$ which is the same as the exact testability for $a \rightarrow f$ *s-a-1* fault.

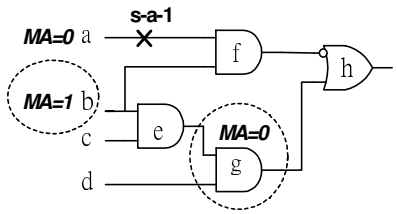


Fig. 5 the example for transitive fanout correlation technique

From the above example, we can further improve the testability results by using the transitive fanout correlation. Our algorithm first obtains the all frontier MAs and then, re-calculates the conditional signal probabilities of all frontier MAs. In this way, the testability analysis of a fault can be improved.

4.2 MA insertion technique

In this subsection, we describe a new technique to improve the accuracy of estimating a multiple signal probabilities so as to improve the testability measurement. Consider to estimate the multiple signal probability $T_F = p(n_1=v_1, n_2=v_2, \ldots, n_k=v_k)$ where $\{n_1=v_1, n_2=v_2, \ldots, n_k=v_k\}$ are all MAs for a fault F. Let node n be an internal node without an MA for fault F. The multiple signal probability T_F can be re-written to

$$T_F = p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k, n = 0) + p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k, n = 1)$$

We would like to mention that ideally, the TAIR algorithm and transitive fanout technique can identify more signal correlations if there are more MAs in a circuit. Since both multiple signal probabilities $p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k, n = 0)$ and $p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k, n = 1)$ have more MA values than $p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k)$, we may obtain more accurate estimation for both multiple signal probabilities than the original one. As a result, using

$$p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k, n = 0) + p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k, n = 1)$$

may obtain more accurate estimation than $p(n_1 = v_1, n_2 = v_2, \ldots, n_k = v_k)$. We use the following example to illustrate this concept.

In Fig. 6, consider to find the testability for fault ***F***, $f \rightarrow h$ stuck-at-1 fault, which have the root MAs of $\{f=0, g=0\}$. The COP's result is $p(f=0)*p(g=0)=3/4*7/8=21/32$. The TAIR algorithm cannot improve the result because there are no new MAs derived in this example. Also, the transitive fanout technique cannot be applied since none of the frontier MAs $\{f=0, g=0\}$ are in the transitive fanout of others. In the MA insertion technique, we first select a node. (How the node is chosen is described later.) In this example, suppose node b is chosen for MA insertion. Then we can re-write the probability by the following.

$T_F = p(f=0,g=0) = p(f=0,g=0, b=0) + p(f=0,g=0, b=1)$

With new assignments of $b=0$ ($b=1$), we can have more accurate estimation using TAIR and the transitive fanout technique.

$p(f=0, g=0, b=0) + p(f=0,g=0, b=1)=p(b=0) + p(a=0, b=1, g=0)=p(b=0) + p(a=0)*p(b=1)*p(g=0|b=1)$

$=1/2 + 1/2*1/2*3/4 = 1/2 + 3/16 = 11/16.$

Then exact testability value 11/*16* for $f \rightarrow h$ stuck-at-1 fault can be obtained.

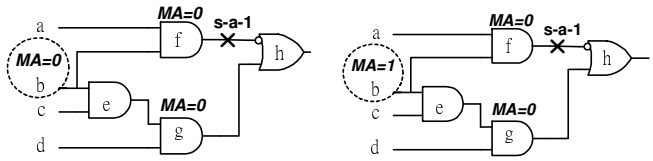


Fig. 6 the example for MA insertion technique

When very accurate testability analysis for some node is essential, one may apply more than one MA insertions. We discuss how two MA insertions can be performed by the following example. Let node n and m be chosen for MA insertions. So the testability T_F can be re-calculated as below equation:

$T_F = p(n_1=v_1, n_2=v_2, \ldots, n_k=v_k)$

$= p(n_1=v_1, n_2=v_2, \ldots, n_k=v_k, n=0, m=0) + p(n_1=v_1, n_2=v_2, \ldots, n_k=v_k, n=0, m=1) + p(n_1=v_1, n_2=v_2, \ldots, n_k=v_k, n=1, m=0) + p(n_1=v_1, n_2=v_2, \ldots, n_k=v_k, n=1, m=1)$

Similarly, each multiple signal probability can have more MAs than the original one so we can get more accurate result for T_F by MA insertions.

We now describe how to select a node for the MA insertion. First, more MAs usually imply more correlations which can be found in TAIR. We intend to select a good candidate node for MA insertion so the MA insertion can imply more MAs. Therefore, our principle of selecting a node for MA insertion is that the MA insertion must cause many new MAs derived. All three methods of selecting MAs described below follow this principle. The first method selects a node that is next to a frontier MA and has the maximum number of fanouts. The second method selects a node that has the maximum number of fanouts. And the last method selects a primary input node that has the maximum number of fanouts.

5 Experimental Results

We have implemented the algorithms for testability analysis and performed experiments on MCNC and ISCAS benchmark circuits. Each circuit for our experiment is first optimize by the script "script.Boolean" in SIS [5] and decomposed into AND and OR gates by using "decomp –good; tech_decompe –a 1000 –o 1000". The all experiments are conducted on UltraSPARC 440MHz machine. In Table 2, the first column gives the name of a circuit. For each circuit, we compute the circuit testability by the formula

$$\frac{1}{|faults|}\sum_{faults}\frac{1}{Pd_i}$$

of [4] for the purpose of performance evaluation. Here, only the testable faults are under consideration. The formula value of "exact" method in the 2th column uses Binary Decision Diagram (BDD) to obtain the exact testability, and the circuits marked with "*" cannot be finished by the BDD method, so they are estimated by random pattern fault simulation with 524,288 (2^{19}) random test patterns. The 3th column shows the TAIR's results. The 4th column shows the results after apply the transitive fanout technique. In the 5th, 6th, and 7th column, we show the results of three different MA insertion techniques where the first one selects the node with the largest number of fanouts next to a frontier MA (MAI1), the second method selects the node with the largest number of fanouts in a circuit (MAI2) and the last method selects the primary input node with the largest number of fanouts (MAI3). Our experimental results demonstrate that the new techniques have successfully improved the testability analysis results. Table 3 shows the CPU run time for each method.

6 Conclusion

In this paper, we first have proved the algorithm of TAIR can be written by some closed-form, secondly, based on the closed-form, we propose the transitive fanout and MA insertion techniques for improving testability analysis. Our experiments show that improvement can be achieved.

References

[1] F. Brglez, "On Testability of Combinational Networks," *Proc. of Int'l Symposium on Circuits and Systems*, pp. 221-225, 1984.

[2] S.C. Chang, W.B. Jone and S.S. Chang, "TAIR: Testability Analysis by Implication Reasoning," *IEEE Trans. on Computer Aided Design* (accept).

[3] K. T. Cheng and C. J. Lin, "Timing Driven Test Point Insertion for Full-Scan and Partial-Scan BIST," *Proc. of Int'l Test Conf.*, pp. 506-514, 1995.

[4] H. C. Tsai, K.T. Cheng, C. J. Lin, S. Bhawmik, "A Hybrid Algorithm for Test Point Selection for Scan-Based BIST," *Proc. Design Automation Conf.*, pp. 478-483, 1997.

[5] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Muragi, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Synthesis," U. C. Berkeley, *Technical Report* UCB/ERL M92/41, May 1992.

Table 2 The circuit testability comparisons.

Circuit	$\frac{1}{\|faults\|}\sum_{faults}\frac{1}{Pd_i}$ (*faults* are those testable faults)					
	Exact	**TAIR**	**TFC**	**MAI1**	**MAI2**	**MAI3**
alu2	77.6	841.5	800.9	399.3	400.9	399.2
alu4	93.2	3530.2	2540.4	1498.7	1363.7	1363.7
apex7	73.3	93.2	80.0	79.2	78.7	78.7
C17	6.3	6.5	6.3	6.3	6.3	6.3
C1908	186.1	218.1	216.7	211.7	216.2	216.2
*C6288	7.1	18.7	17.8	18.2	17.9	17.9
*C7552	2964.6	121073.5	104459.1	103839.4	97414.4	97416.7
*i10	2111.2	10009.8	5696.4	5455.5	5015.8	5018.6
i6	8.8	8.8	8.8	8.8	8.8	8.8
i8	44.1	40.7	42.3	49.3	49.5	42.4
sct	109.0	69.3	87.3	89.0	86.3	86.0
t481	14515.1	12963.4	13555.9	13860.9	13703.0	13888.6
too_large	234498.8	34361.7	73906.3	154264.1	153856.4	153856.4
ttt2	115.2	89.2	117.2	98.1	97.0	97.2
x1	661.0	841.9	729.6	705.9	732.1	732.1
Average	**1**	**0.721**	**0.792**	**1.098**	**1.069**	**1.070**

Table 3 The run time comparions.

Circuit	Run time (*sec.*)					
	Exact	**TAIR**	**TFC**	**MAI1**	**MAI2**	**MAI3**
alu2	27.69	0.71	2.31	3.90	4.17	4.10
alu4	107.87	2.10	7.89	13.61	14.44	14.16
apex7	10.14	0.11	0.80	1.32	1.63	1.58
C17	0.04	0.00	0.01	0.01	0.00	0.01
C1908	550.57	0.76	3.25	5.69	6.15	6.00
*C6288	31883.01	17.25	124.14	209.24	238.64	231.88
*C7552	17545.81	3.59	65.95	120.93	139.46	132.48
*i10	17180.25	4.42	61.99	113.18	124.54	120.93
i6	31.52	0.55	3.09	4.60	6.16	5.99
i8	3105.67	1.92	13.19	20.17	24.29	25.51
sct	1.06	0.04	0.11	0.19	0.19	0.19
t481	100.36	2.17	6.75	10.80	10.61	11.18
too_large	90.22	0.40	2.06	3.80	4.10	3.98
ttt2	8.37	0.12	0.70	1.16	1.39	1.35
x1	14.83	0.15	1.04	1.89	2.06	2.00
Average	**2060.6**	**1**	**8.6**	**14.9**	**16.9**	**16.4**