Fast Static Compaction of Test Sequences using Implications and Greedy Search

J. Raik, A. Jutman, R. Ubar

Tallinn Technical Univ., Raja 15, 12618 Tallinn, Estonia, E-mail: {jaan|artur|raiub}@pld.ttu.ee

Abstract

Current paper presents a new technique for static compaction of sequential circuit tests that are divided into independent test sequences. The technique implements effective representation of fault matrices by weighted bipartite graphs. The approach contains a preprocessing step for determining the set of essential vectors. Subsequently, implications and a greedy search algorithm is applied. The proposed method offers significantly faster performance in terms of run times than earlier, genetic algorithm based methods. Moreover, the average compaction provided by current method is better.

1 Introduction

Minimization of the number of patterns in a test set is an essential problem for the chip manufacturer, who faces the test of millions of units per annum [1]. The time required to test a chip by the ATE is directly proportional with the length of the test sequence. Therefore, the number of patterns in a test set is an important parameter when speaking of test pattern generation. Minimization of this number is refered to as test sequence compaction.

There exist two types of test compaction techniques: static and dynamic. In static compaction [2, 3, 4, 5], a test sequence is generated and subsequently attempts are made to shorten it without reducing its fault coverage. The main advantage of the static techniques is that they are independent of the adopted ATPG tool. Dynamic test set minimization [5, 6, 7], on the other hand, is performed at the time when tests are being generated. This requires modification of the test generation algorithm itself in order to make it generate shorter sequences.

Many of the works in the field of static compaction [2, 4] consider the case, where there is a single test sequence that we are trying to minimize by removing some patterns from it. This requires iterative fault simulation during the compaction process in order to check that the fault coverage has not decreased. Thus the run times are very long.

Faster approach has been proposed in [3] and in [5]. The technique in [5] requires keeping track of the internal state of the circuit. In [3] the whole test set is divided into independent test sequences separated by global reset and fault simulation is performed only once, prior to compaction. In addition, in [3] a set of benchmarks [9] consisting of 103 fault matrices of ISCAS89 circuits tested by three different ATPG tools [10, 11, 12] were made publicly available.

In this paper we target the above mentioned case of static compatcion where the test set is divided into test sequences. We propose a technique that uses an effective representation of fault matrices by weighted bipartite graph models which provide for a more compact means of describing the test sets than traditional matrix representations. It contains a preprocessing step for determining the set of essential vectors of the test sequences. This step considerably reduces the search space for the compaction algorithm. Subsequent to preprocessing, search space pruning and a greedy search algorithm are applied in order to compact the test set.

2 Model representation and basic definitions

Consider the test set example shown in Figure 1 that consists of three test sequences s_1, s_2 and s_3, respectively. Sequence s_1 consists of four test vectors covering fault f_2 at the third vector and f_1 at the fourth vector. Sequence s_2 consists of three test vectors covering f_1 at the first vector and f_3 at the third vector. Finally, sequence s_3 consists of four test vectors covering f_2 at the first vector, f_3 at the second vector and f_4 at the fourth vector.

Initial test length of this test set is 11 vectors. It can be found that the optimal solution for the static compaction problem is selecting sequence s_3 and the first vector from sequence s_2. Hence, the length of the optimal compacted test set will be 5 vectors.

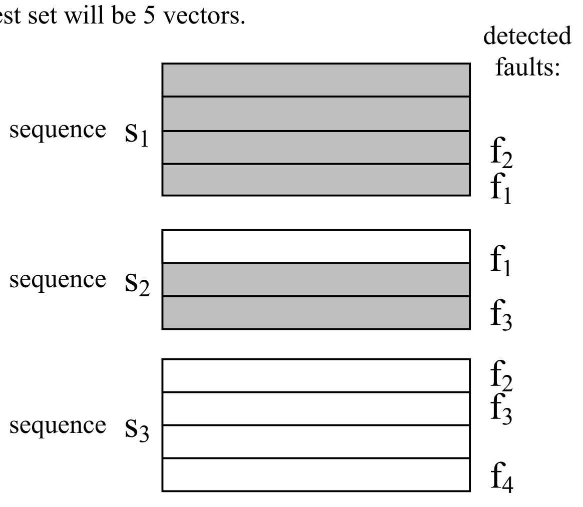

Figure 1. Test set example

In current implementation the test set information is represented by a model of weighted bipartite graphs. The motivation for this is the fact that bipartite graph models generally provide for a much more compact means of describing the test sets than matrix representations. We use a *weighted bipartite graph* $G_{n,m}$, where the first part of the graph consists of n vertices that correspond to test sequences s_i and the remaining part has m vertices corresponding to the faults f_j detected by the test set. There exists an edge connecting vertices s_i and f_j iff sequence s_i covers fault f_j. Edge $e = <s_i, f_j>$ is labeled by an integer c, $c = w(e)$, where fault f_j is covered at the c-th vector of test sequence s_i.

The weighted bipartite graph representation for the test set in Fig. 1 is shown in Fig. 2.

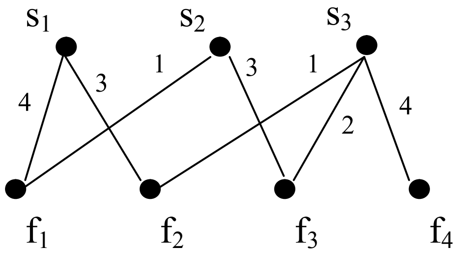

Figure 2. Weighted bipartite graph

However, for the sake of simplicity, in the following algorithm descriptions we will represent the test set by a matrix, where the rows correspond to sequences and columns correspond to faults [3]. The fault matrix representation for the test set in Fig. 1 is shown below.

	f_1	f_2	f_3	f_4
s_1	4	3	0	0
s_2	1	0	3	0
s_3	0	1	2	4

In this type of descriptions test set T consisting of n faults and m test sequences can be viewed as a matrix

$$T = \begin{pmatrix} t_{f_1,s_1} & t_{f_2,s_1} & \cdots & t_{f_n,s_1} \\ t_{f_1,s_2} & t_{f_2,s_2} & \cdots & t_{f_n,s_2} \\ \cdots & \cdots & \cdots & \cdots \\ t_{f_1,s_m} & t_{f_2,s_m} & \cdots & t_{f_n,s_m} \end{pmatrix},$$

where $t_{si,fj}$ is equal to k if sequence s_i covers fault f_j at the k-th vector and zero if sequence s_i does not cover fault f_j.

If we select k vectors from sequence s_i then all the faults $\{f_j : k \geq t_{si,fj} > 0\}$ are said to be covered by these vectors. In our algorithm we remove the columns corresponding to the covered faults from matrix T. In addition, we must subtract k from all the non-zero elements $t_{si,fj}$ of the row corresponding to the sequence s_i.

Our task is to cover all the faults (i.e. columns of matrix T) by selecting the minimal number of vectors. As it was shown in [3], this task belongs to the class of NP-complete problems.

3 Compaction algorithm

A simple pre-processing step of detecting *essential vectors* from the test sequences is applied at the beginning of the compaction algorithm. If fault f_j is detected by the k-th vector of test sequence s_i and is not detected by any other sequence then k first vectors of sequence s_i are called essential. After selecting the essential vectors we remove them from the test sequences. In addition we remove the columns corresponding to faults covered by these vectors from matrix T. This simple pre-processing step allows to significantly reduce the search space for the static compaction algorithm.

In addition to selecting the set of essential patterns, two other types of implications are made. These are collapsing of equivalent faults and removing subrows, respectively. During *collapsing of equivalent faults*, column f_a will be removed from matrix T if there exists another column f_b, where $\overset{m}{\underset{i=1}{\forall}} t_{f_a,s_i} = t_{f_b,s_i}$.

In other words, if we have multiple identical columns we will unite them into a single one.

Another type of implications is *removing subrows*. A row corresponding to sequence s_b is said to be a subrow of s_a iff $\overset{n}{\underset{j=1}{\forall}} t_{s_b,f_j} \neq 0 \Rightarrow t_{s_a,f_j} \neq 0,\ t_{s_a,f_j} \leq t_{s_b,f_j}$.

Current technique applies above described implications as far as possible. When it encounters a selection between alternative solutions, it switches to a greedy algorithm [13]. The greedy selection function implemented in current technique is described in the following. Let us denote by *Minrange*(f_j) the minimal number of vectors that has to be selected from any test sequence in order to detect a fault f_j. Let *Maxrange* be the maximum *Minrange*(f_j) of all the faults.

$$Minrange(f_j) = \min_{i=1}^{m}\{t_{s_i,f_j} : t_{s_i,f_j} \neq 0\}.$$

$$Maxrange = \max_{j=1}^{n}\{Minrange(f_j)\}.$$

The selection function selects *Maxrange* vectors from the corresponding test sequence. If there exist multiple maximal *Minrange*(f_j) values then the algorithm prefers the sequences that detect more faults in *Maxrange* first vectors. In the following the description of the algorithm for static compaction is presented:

```
Select essential vectors.
Remove the faults covered by these vectors.
While exist uncovered faults
{
  Remove subrows.
  Collaps equivalent faults.
  If new essential vectors appeared then
    Select essential vectors.
  Else
    Select vectors by greedy selection.
  Endif
  Remove the faults covered by selected vectors.
}
```

4 Detecting lower bounds and global optima

Since the algorithm described in the previous section is using implications, it allows it to calculate the lower bounds for the static compaction task. The meaning of the lower bounds is that they show that it is not possible to compact the test set to contain fewer vectors than the bound. Moreover, in the cases where the result found by the algorithm equals the lower bound we have proved that this result is the global optimum.

In current approach, the lower bound is determined by our technique with the number of vectors selected during the implications up to the first greedy selection, including the vectors chosen by that selection. The first greedy selection can be included due to the fact that, obviously, it represents the minimal number of vectors that are necessary in order to cover a previously uncovered fault f_j. All the alternative combinations of selecting vectors for covering f_j must always result in a greater or equal number of vectors.

5 Experimental results

Both, the experiments of current approach and the comparative experiments of [3] were run on SUN SPARC 5 computer. We used the test set benchmarks that can be downloaded from [9]. The benchmarks include test sets for three different ATPG tools: GATTO [10], HITEC [11] and SYMBAT [12]. GATTO is a genetic algorithm based ATPG, HITEC is a deterministic gate-level ATPG and SYMBAT is based on symbolic test generation techniques.

Experiments show that current technique offers 16,7 - 294 times shorter CPU times and is in average 74,3 times faster than the method implemented in [3]. The run time statistics for test set benchmarks of different ATPG tools are presented in Table 1.

Table 1. Speed-up in comparison to [3]

Speed-up, times	Average	Max	Min
GATTO test sets	77.1	218.2	16.7
HITEC test sets	83.1	294.1	20.0
SYMBAT sets	54.3	200.0	22.0

Figure 3 and Table 2 show the average compaction achieved by current approach in comparison to compaction of [3] and the theoretical lower bound calculated by current technique. As it can be seen from the Table, compaction of this technique is in average better than the one of [3] for all the ATPG test sets. In the case of HITEC and SYMBAT the technique allows much closer to optimum results than reported in [3]. These results are very near to the calculated lower bound.

Table 2. Average compaction of test sequences

	GA [3]	curr. appr.	bound
GATTO test sets	49.86 %	50.06 %	51.28 %
HITEC test sets	43.30 %	43.98 %	44.15 %
SYMBAT sets	38.30 %	38.33 %	38.35 %

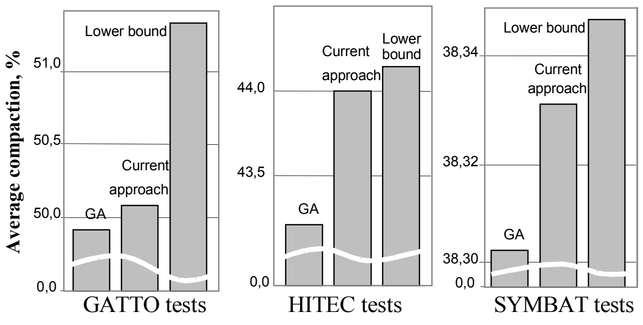

Figure 3. Comparison of average compaction

Conclusions

In this paper, the case of static compaction where the test set is divided into test sequences is considered. A new technique is proposed that allows better compaction than any previously published method belonging to this particular class. Moreover, experiments show that the technique offers 16.7-294 times shorter CPU times and is in average 74.3 times faster than the method in [3].

Unlike previously published approaches, this technique is capable of detecting and proving globally optimal results for most of the benchmark test sets in [9]. Global optima are proved for 50 % of GATTO test sets, 82.5 % of HITEC test sets and 91.3 % of SYMBAT test sets using this approach.

References

[1] K. M. Thomson, "Intel and the myths of test". *IEEE Design & Test of Computers*, Spring 1996, pp. 79-81.

[2] I. Pomeranz, S. M. Reddy, "On static compaction of test sequences …". *Proc. Design Automation Conf.*, 1996.

[3] F. Corno, et al., "New static compaction techniques of test sequences for sequential circuits". *ED&TC*, 1997.

[4] I. Pomeranz, S. M. Reddy, "Vector restoration based static compaction …". *Proc. of ICCD*, 1997.

[5] M.S. Hsiao et al., "Fast static compaction algorithms …". *IEEE Trans. Comp.*, Vol. 48, No. 3, 1999.

[6] P. Goel, B. C. Rosales, "Test generation and dynamic compaction of tests". *Digest of Papers Test Conf.*, 1979.

[7] I. Pomeranz, S. M. Reddy, "Dynamic test compaction …". *IEEE Fault Tolerant Computing Symp.*, 1996.

[8] E. M. Rudnick, J. H. Patel, "Putting the squeeze on test sequences". *Proc. of ITC*, 1997, pp. 723-732.

[9] URL: http://www.cad.polito.it

[10] F. Corno, et al., "GATTO: A genetic algorithm for ATPG … ", *IEEE Trans. CAD*, Aug. 1996.

[11] T.M. Niermann, J.H. Patel, "HITEC: A test generation package for sequential circuits", *Proc. of EDAC*, 1991.

[12] G. Cabodi, F. Corno, P. Prinetto, M. Sonza Reorda, "Symbat's user guide", Politecnico di Torino, 1993.

[13] J. Edmonds, "Matroids and the greedy algorithm". *Mathematical Programming*, Vol. 1, 1971, pp. 127-136.